Universidad de Alcalá
Escuela Politécnica Superior

INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

SISTEMA DE CONTROL DE ACCESO INTEGRAL A UN PARKING CON BARRERA

Autor: Sebastian Iosif Catrina

Tutor : Miguel Ángel García Garrido

2022-2023

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Ingeniería Informática

Trabajo Fin de Grado

SISTEMA DE CONTROL DE ACCESO INTEGRAL A UN PARKING CON BARRERA

Autor: Sebastian Iosif Catrina

Tutor/es: Miguel Ángel García Garrido

TRIBUNAL:

Presidente: Enrique Santiso Gómez

Vocal 1º: Juan Manuel Miguel Jiménez

Vocal 2º: Miguel Ángel García Garrido

FECHA: 23 de Septiembre del 2022

Resumen

Este proyecto tiene como objetivo la implementación de un sistema de control de acceso mediante identificación de matrículas, aplicado al aparcamiento de la Escuela Politécnica Superior.

El proyecto consta de varias fases. La primera es la encargada de realizar la captación de matrículas, la segunda será la creación de un servidor central que trabajará los datos y ofrecerá una interfaz para que el usuario acceda y modifique sus datos a través de internet. La última parte se centrará en la creación de una aplicación móvil que permite consultar los datos del alumno y recibir notificaciones cuando se accede al recinto.

Palabras Clave

ANPR, MQTT, Servlet, PHP, SQL

Abstract

The goal of this project is creating a system that will control the access to a parking using License Plate Recognition, applied to the parking of the EPS.

The project will be divided in three phases, the first one will implement the license recognition. The second one will create the server that will process all the data and will offer a webpage to the user. In the last one a mobile application, for Android, will be created to enable the user to retrieve his data and to get notifications when any of his cars arrives, informing him about his next class.

Keywords

ANPR, MQTT, Servlet, PHP, SQL

Índice

Índice de Figuras

1. Introducción

Se ha creado un sistema de control de acceso a un aparcamiento con barrera, compuesto por varios módulos que se comunican entre sí permitiendo guardar los accesos y notificar los próximos eventos que tendrá el usuario (asignaturas, exámenes…). Este proyecto se centra en crear una mejora del sistema de acceso existente actualmente en la Escuela Politécnica Superior de la Universidad de Alcalá.

La primera parte del proyecto se centró en investigar el proceso de adquisición de datos haciendo uso de una cámara web, que será usada para captar las matrículas de los vehículos y así poder informar al sistema acerca del vehículo que está en la berrera. Una vez el sistema sea capaz de detectar lo que hay en la barrera, se implementó un servidor conectado a una base de datos en la que se registrará la información de los vehículos asociados a cada miembro de la UAH (alumnos, profesores, personal de seguridad…). Esto se empleará para conocer quién accede o abandona la instalación en cada instante y, en caso de que no se reconozca la matrícula, el vehículo se registrará como invitado (se permite el acceso ya que es lo equivalente al llamar al timbre que se emplea actualmente, el cambio será que quedan guardados los datos del vehículo que ha accedido).

La segunda parte se centró en el diseño de una página web, que permite a los integrantes actualizar la información de sus vehículos, consultar el historial de accesos obtener información acerca de sus futuras clases. Ésta está conectada a la base de datos para poder crear las entradas, también cuenta con un sistema de login que permite que los alumnos accedan con su correo universitario. Una posibilidad sería hacer uso del correo universitario para identificar en que asignaturas está matriculado el alumno, pero esto supondría tener que acceder a alguna base de datos de la universidad, así que se ha decidido ofrecer una lista con todas las asignaturas y que el alumno pueda elegir aquellas en las que este matriculado, de modo que se le notificarán los futuros eventos de las que este seleccione.

El servidor puede ser accedido tanto mediante la página web o mediante una aplicación móvil. En ambos casos la interfaz se modificará dependiendo del rol de la persona que acceda, en caso de que sea un administrador podrá consultar los accesos de todos los demás roles junto con sus vehículos o asignaturas matriculadas. En caso de que sea un alumno únicamente podrá modificar sus vehículos, consultar su historial de accesos, ver sus eventos próximos o añadir eventos (tutorías, quedadas para hacer trabajos…). Los profesores, aparte de lo que pueden hacer los alumnos, pueden modificar los horarios de las asignaturas de las que responden o añadir eventos para los alumnos matriculados.

1.1 Diagrama general

Se ha separado el trabajo en tres bloques principales, el primero está formado a su vez por dos partes, la cámara y el servidor por un lado y el Arduino con el sensor de ultrasonido por otro.

Como se observa en la Figura 1, el primero de ellos sería el encargado de obtener las matrículas. Este bloque está formado por un módulo que medirá la distancia del coche hasta este y actuará como trigger, para que comience el proceso de detección únicamente cuando hay algún vehículo cerca y el ordenador encargado de realizar la detección de la matricula, obteniendo la imagen mediante la cámara conectada, procesándola para obtener una captura más clara de la matricula y aplicando OCR.

Por otra parte, existirá un servidor encargado de recibir y procesar los mensajes de las barreras y decidir que vehículos tienen acceso. También guardará la página web donde se pueden consultar los datos y el servidor PHP del que los dispositivos móviles cargarán los datos. Este recibirá los mensajes enviados por la barrera una vez se detecte un vehículo, los procesará para consultar el tipo de barrera (si es de entrada o de salida) y enviará una notificación al usuario que tenga registrado el vehículo acerca de su próxima clase, si existe.

El ultimo bloque será el encargado de gestionar la aplicación para los dispositivos móviles, que disponen de aplicación propia en el caso de Android. Los móviles recibirán notificaciones cuando sus vehículos entren o salgan del recinto, informándoles que sus próximas asignaturas, en caso de que las haya.

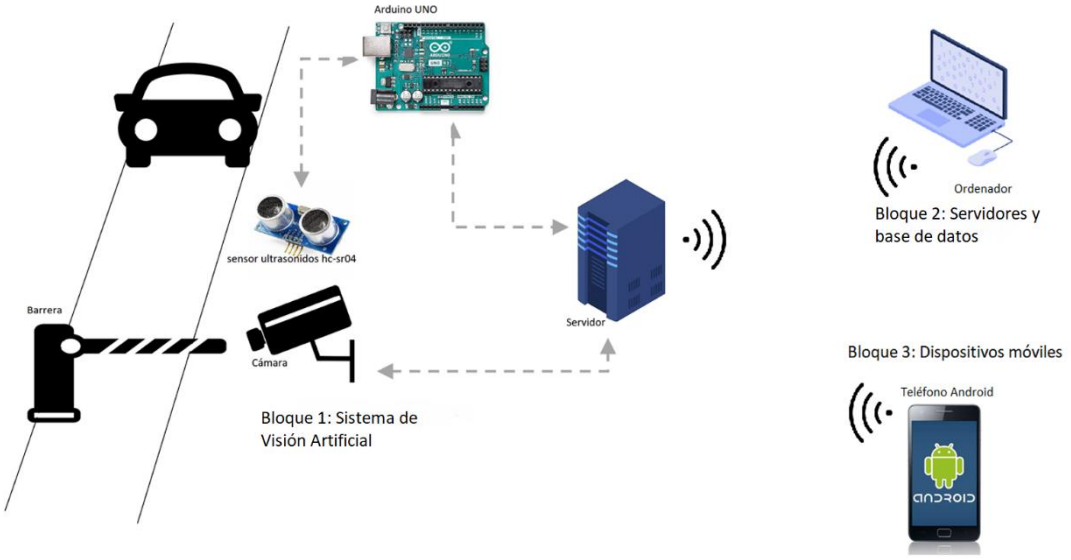

Figura 1: Esquema del sistema final

2. Objetivos

El objetivo de este TFG es implementar un sistema de acceso haciendo que controle la barrera mediante el uso de una cámara para detectar las matrículas.

Para poder lograr esto se ha separado el trabajo en múltiples objetivos más pequeños, estos serán:

1. Implementar el reconocimiento de la matricula.
    a. Obtener las imágenes de la cámara.
    b. Procesar la imagen y obtener caracteres.
    c. Enviar la matricula al servidor.
2. Crear una base de datos para guardar accesos, clases y los datos del usuario.
    a. Crear base de datos.
    b. Crear tablas.
    c. Crear relaciones.
3. Crear servidor MQTT.
4. Crear servidor web.
    a. Crear página web para el usuario.
    b. Implementar procesado para los datos.
    c. Establecer conexión con la base de datos.
    d. Establecer conexión con el servidor MQTT.
5. Crear aplicación Android.
    a. Diseñar la aplicación.
    b. Conectar con la base de datos.
    c. Conectar con MQTT.
    d. Implementar servicio de notificaciones.

3. Trabajo desarrollado

El trabajo se ha dividido en varios bloques separados e independientes, ya que se pueden realizar modificaciones en cada uno de ellos sin influir en el funcionamiento de los demás. Estos bloques se comunicarán entre si haciendo uso de mensajes MQTT, de servlets y de consultas PHP.

3.1 Bloque 1: Sistema de Visión Artificial

Este primer bloque será el encargado de captar las matrículas de los vehículos que se aproximen, mantener los servidores y procesar las peticiones, gestionando la base de datos. Para poder realizar esto se ha dividido a su vez en varios componentes. Consistirá en un componente encarado de medir las distancias hasta el vehículo, El principal de estos está formado por el ordenador y una cámara, que se usará para obtener las imágenes. Este estará en modo de espera hasta que se reciba una señal (trigger) indicando que se ha acercado un vehículo y puede comenzar el proceso de reconocimiento de matrículas. Este proceso comenzará tratando de corregir la perspectiva de la imagen, aplicando homografía. Después hará uso de OCR para tratar de obtener los caracteres de la imagen.

El procesamiento de la imagen se denomina visión artificial, esta es la ciencia que se encarga de analizar imágenes reales para tratar de obtener datos de ellas empleando máquinas. Esto se logra descomponiendo las imágenes en pixeles y analizándolos posteriormente. Para poder hacer esto se requiere un elemento que capture los datos, en general una cámara, posteriormente los datos se envían a un computador para ser procesados. Además, se suele hacer uso de otros elementos para complementar la toma de imágenes tales como elementos de iluminación que mejoran la calidad de las imágenes o sensores que indican cuando se deben capturar estas.

Algunos sectores en los que esta se usa son:

Automoción: La mayoría de los vehículos actuales cuentan con sistemas de visión que se emplean como asistentes a la conducción, desde el reconocimiento de señales de tráfico, de carriles para mantener el vehículo entre ellos y, sobre todo, en los sistemas de conducción autónomos.

Reconocimiento facial: Se emplea tanto en los sistemas de desbloqueo de los móviles, reconocer al propietario de la cuenta en los cajeros automáticos o detección de personas en la calle. Un avance reciente en este ámbito han sido las nuevas tiendas inteligentes, que reconocen a los clientes al entrar y les cobran los artículos tan pronto como los cogen de la estantería, haciendo que estos no se tengan que preocupar por pagarlos y evitando la formación de colas en los cajeros.

Realidad mixta: Creación de dispositivos que analizan el entorno para integrarlo con elementos virtuales.

Actualmente existen sistemas completos que realizan este trabajo, llamadas cámaras LPR/ANPR (License Plate Recognition/ Automatic Number Plate Recognition), preparadas para poder reconocer vehículos a suficiente distancia y que funcionan incluso en condiciones de baja luminosidad. Estas están preparadas para detectar correctamente las matrículas a una determinada distancia, con un determinado ángulo vertical y horizontal. Estas se pueden encontrar desde 100€, de modo que la parte que realmente

incrementa el presupuesto es el software encargado de tratar las imágenes. Se han encontrado varios proyectos en los que se trata el tema de la visión artificial, así que nos hemos inspirado en esos y contemplado varios análisis previos que realizaron, observando las funciones que emplean y comparando el rendimiento de estas, para así poder escoger la solución que ofrezca el mejor equilibro entre tiempo de procesado y calidad de la detección.

Se ha decidido hacer uso de OpenCV para el recorte de la matrícula y de TesseractOCR para el reconocimiento del texto.

3.1.1 Detección de distancia

Para esta parte se ha hecho uso de una placa Arduino y de un sensor de ultrasonidos HC-SR04, ya que esta parte requiere de pocos recursos, se podría hacer uso de cualquier placa del estilo, en este proyecto se hará uso de una Arduino UNO (Figura 2):

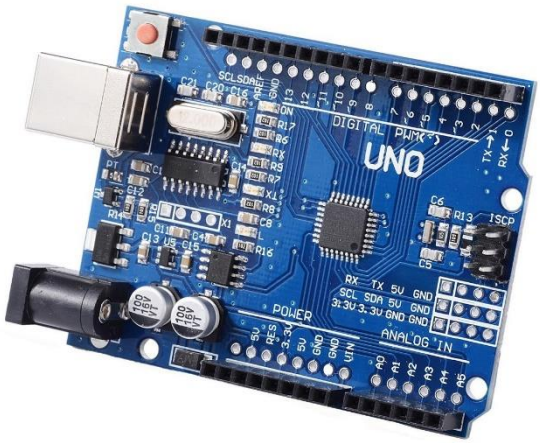

Figura 2: Arduino UNO R3

Esta placa posee las siguientes características:

- Microcontrolador: ATmega328P
- Conector USB: USB-B
- Velocidad de reloj: 16MHz
- Voltaje de trabajo: 5V
- Voltaje de entrada: 7.5 a 12V
- Pinout: 14 pines digitales (6 PWM) y 6 pines analógicos
- Memoria: 32 KB Flash (0,5 para bootloader), 2KB RAM y 1KB Eeprom

El otro elemento que participa en este módulo será el sensor de ultrasonidos (Figura 3):

Figura 3: Sensor Ultrasonido HC-SR04

Características del sensor:

- Alimentación: 5V
- Interfaz de cuatro hilos (vcc.trigger,echo,GND)
- Rango de medición teórico: 2cm a 400cm
- Corriente de alimentación: 1.5mA
- Frecuencia del pulso: 40Khz
- Apertura del pulso: 15º
- Señal de disparo: 10us

Funcionamiento del sensor:

Para detectar las distancias, el sensor emite pulsaciones acústicas (inaudibles para los humanos) a través del aire. Estas señales rebotan en los objetos que haya en frente y rebotan de vuelta al sensor, midiendo este el tiempo que tarda la onda en ir y volver.

Posteriormente se tiene en cuenta la velocidad a la que viaja el sonido en el aire para calcular la distancia en función del tiempo, haciendo uso de la fórmula que se ve en la Ecuación 1:

$$343\frac{m}{s} \cdot 100\frac{cm}{m} \cdot \frac{1}{1000000}\frac{s}{\mu s} = \frac{1}{29.2}\frac{cm}{\mu s} \qquad (1)$$

La velocidad del sonido en el aire es de 343 m/s, de modo que se puede calcular que el tiempo empleado en recorrer 1cm es de 29.2 microsegundos. Ya que la medida que ofrece el sensor está en microsegundos, bastará con dividir el resultado entre 29.2 * 2(ya que es la distancia ida y vuelta) para obtener la distancia.

Tras realizar varias pruebas con este sensor se ha observado que la distancia de medición real difiere bastante de la especificada, no se han podido detectar objetos a más de 2m de distancia. Esto puede darse debido a una baja calidad de fabricación del sensor o a que no se están realizando las pruebas en un ambiente idóneo. Por esto se ha tenido que separar el bloque que medirá la distancia (Figura 4) de donde ira colocada la cámara, ya que si no estaría demasiado cerca y no se obtendría bien la imagen.

Esquema de montaje del sistema:

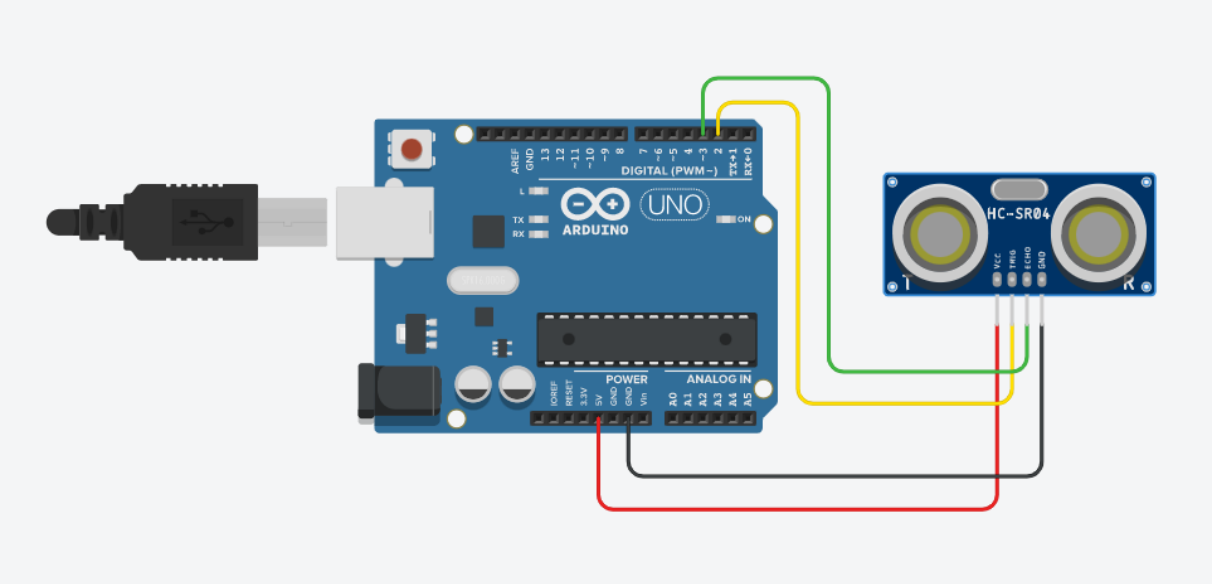

Figura 4: Esquema montaje sensor distancia

Funcionalidad:

El sensor de ultrasonidos realizara una medición cada segundo y comprobará que la distancia no sea inferior a 200 cm. En caso de que lo sea la placa actuara como un trigger para el programa encargado de detectar las matrículas, para hacer esto simulará la pulsación de la tecla ‘f’ del teclado para comunicarle al ordenador que puede realizar mediciones. En caso contrario seguirá comprobando a la espera de que la distancia disminuya.

Para programar la placa se ha hecho uso del IDE de Arduino, este es una aplicación multiplataforma que está escrita en java. Es el software más utilizado a la hora de programar controladores Arduino, no solo porque sea gratis y open-source, sino porque es compatible con todos los modelos de placas, hasta las no oficiales de Arduino y se puede importar una gran cantidad de librerías directamente desde el IDE.

Se usará para compilar y cargar el código a la placa.

En la Ilustración 4 se puede observar que la interfaz es bastante sencilla y limpia, está formada por la barra de herramientas superior, donde tenemos los ajustes de la aplicación, la opción para compilar/subir el programa y la opción para abrir el monitor serie, donde podremos ver los mensajes que nos envía la placa.

También vemos la consola de texto, donde colocaremos el código y un pequeño terminal en la parte baja que nos devolverá información acerca de la compilación, subidas a la placa o errores.

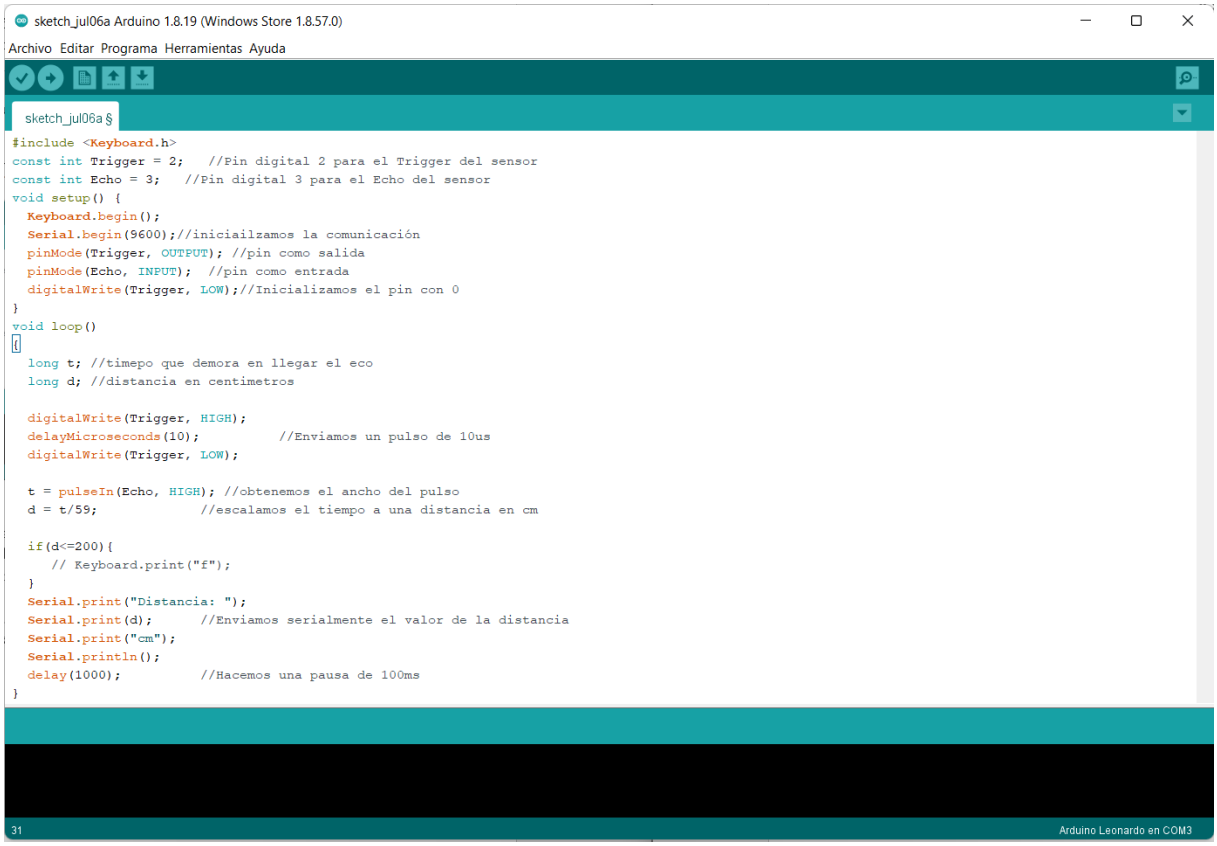

Figura 5: Arduino IDE

3.1.2 Captura de la imagen

Al activarse el trigger porque se ha acercado un vehículo, da comienzo el proceso de captación de la imagen. Este realizará una serie de capturas para ir comprobando las imágenes captadas desde distintas distancias, se analizarán y, en caso de que se haya reconocido y el vehículo no haya pasado anteriormente, se notifica el acceso. Para esto se seguirá el flujo de la figura 6.

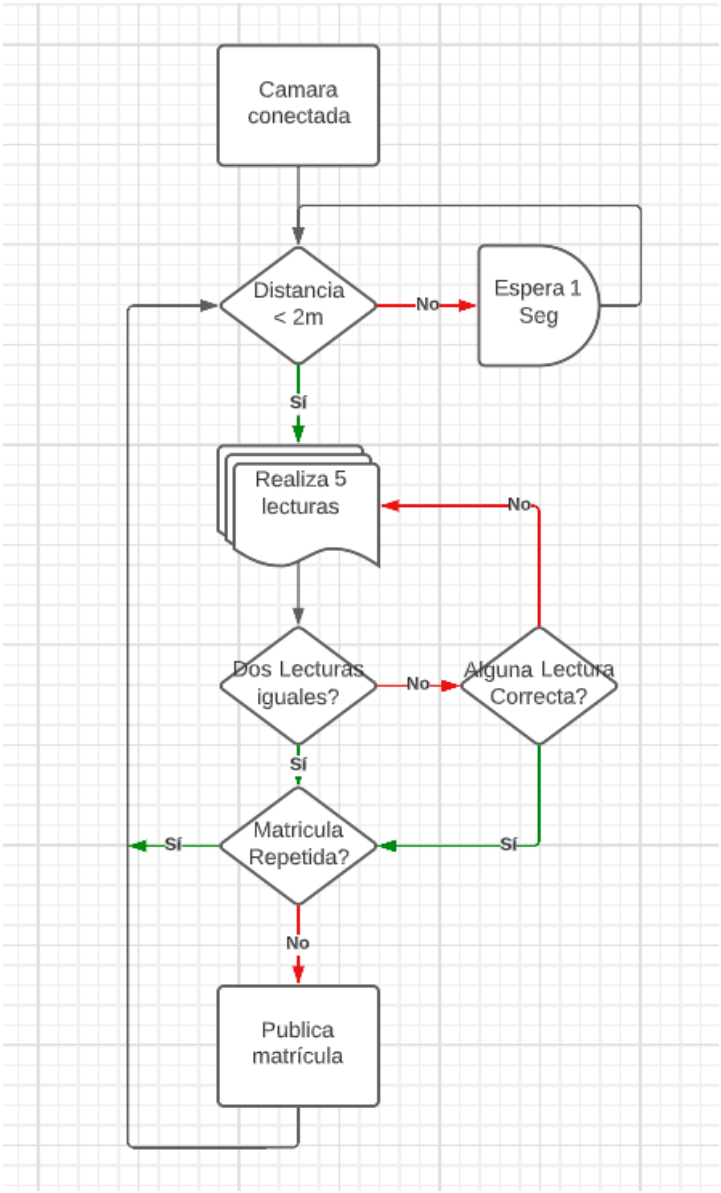

Figura 6: Lectura de Matrícula

Una vez se dé el aviso de que hay algún objeto cerca, comienza la fase de detección de matrículas. Para ello se hará uso de una cámara web conectada al ordenador. En este proyecto se usa la Logitech C270 (Figura 7). Una cámara que se ha notado algo insuficiente a la hora de tomar capturas en condiciones de baja luz o en caso de que el vehículo se acerque a altas velocidades, de modo que este modelo no se podría utilizar en un prototipo real, pero para las pruebas que se han realizado ha funcionado de forma correcta.

Figura 7: Logitech C270

Especificaciones del fabricante:

Descripción: Videoconferencias HD 720p básicas

Resolución/FPS: HD 720p/30 FPS

Campo de visión: 55º

Enfoque automático: No

Micrófonos: 1

Conexión: USB-A

Longitud del Cable: 1.5m

Esta cámara permite, en condiciones de buena iluminación, tomar imágenes lo suficientemente buenas como para poder identificar la matricula a distancias inferiores a 5m.

Para poder tomar las imágenes se ha hecho uso de OpenCV un software libre de visión artificial desarrollada por Intel. Su nombre viene de Open Computer Vision y es una de las bibliotecas más populares para visión artificial.

La biblioteca tiene más de 2500 algoritmos optimizados que se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D de cámaras estéreo, unir imágenes para producir una en alta resolución, encontrar imágenes similares de una base de datos, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer el escenario y establecer marcadores para superponerlo con la realidad aumentada, etc... OpenCV tiene más de 47 mil miembros en la comunidad y el número estimado de descargas superan los 18 millones. La biblioteca se utiliza ampliamente en empresas, grupos de investigación y organismos del gobierno.

Se ha elegido PyCharm para desarrollar la aplicación en Python. Es un programa multiplataforma que permite crear un entorno virtual para cada aplicación, que posee un terminal integrado desde el que poder instalar características, en la Figura 8 se observa su interfaz.

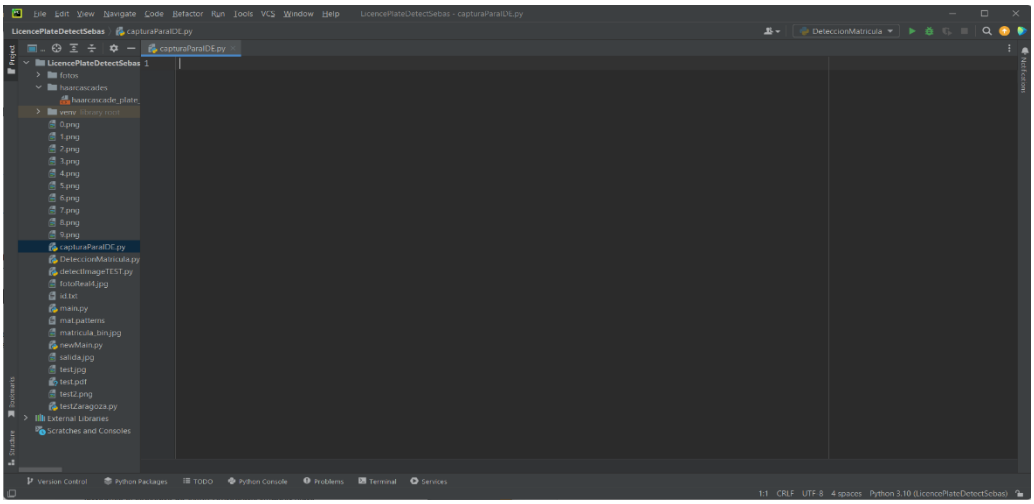

Figura 8: PyCharm Community Edition

Este programa se configurará para estar continuamente con la cámara activa y tomará capturas únicamente cuando el trigger le indique que hay un vehículo cerca. También había que conectarse al bróker al inicio del programa para que se puedan enviar notificaciones al detectar un acceso. Para esto se ha hecho uso del código de la Figura 9.

```
def iniciar_ejecucion(num_fotos):
    vid = cv2.VideoCapture(0)
    id_cam = "1"
    broker_address = "localhost"
    client = mqtt.Client('test1')  # Creación del cliente
    client.connect(broker_address)
    topic = "idCam=" + id_cam
    detectado_anterior = ""
```

Figura 9: Iniciar cámara y conectar al bróker

El ID de la cámara se emplea para distinguir las barreras en caso de tener varias y, de igual modo, saber si son de entrada o de salida.

Una vez cargado y conectado, el sistema entrará en un bucle infinito esperando a que el usuario o, en nuestro caso, el Arduino, pulse la tecla f (Figura 10) indicando que se ha activado el trigger. Esto iniciará el proceso de lectura por parte del ordenador.

```
    while True:
        ret, frame = vid.read()
        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == ord('f'):
            i = 0
            matriculas = []
```

Figura 10: Inicio del bucle y espera para tomar capturas

Se inicia un bucle para tomar el número de fotos deseado, que se ha configurado a 5, esto se hace para capturar distintas imágenes mientras el vehículo se va acercado y así evitar posibles problemas tales como reflejos, sombras u objetos que tapen la matrícula. Para esto se indica el número de capturas deseadas en la llamada a la función, haciendo uso de una toma de imágenes más redundante se logra que el sistema sea más robusto. Para cada iteración se toma una foto y se guarda en disco.

Después se llamará al proceso encargado de obtener el texto de la imagen, en el primer paso se llama a la función que tratará de corregir la perspectiva, en caso de este proceso no sea capaz de obtener la matrícula, se volverá a aplicar, pero sin hacer uso de la corrección de perspectiva. Entre iteraciones hay una pausa de 300 milisegundos para que el vehículo se siga moviendo y, de este modo, poder tomar capturas desde varios ángulos. Este proceso se muestra en la Figura 11.

```
for i in range(num_fotos):
    result, image = vid.read()
    if result:
        cv2.imwrite(str(i) + ".png", image)
        loc = str(i+1) + ".jpg"
        met = obtener_texto(loc)
        matriculas.append(met)
        print(met)
    else:
        print("No image detected. Please! try again")
    time.sleep(0.3)
```

Figura 11: Toma de imágenes

En las Figuras 12,13,14 y 15 se pueden observar algunas capturas tomadas en condiciones de buena iluminación:

Figura 12: Captura a 5,5m

Figura 13: Captura a 4,5m

Figura 14: Captura a 3m

Figura 15: Captura a 2m

En las Figuras 16, 17 y 18 se observan capturas tomadas en condiciones de baja iluminación:

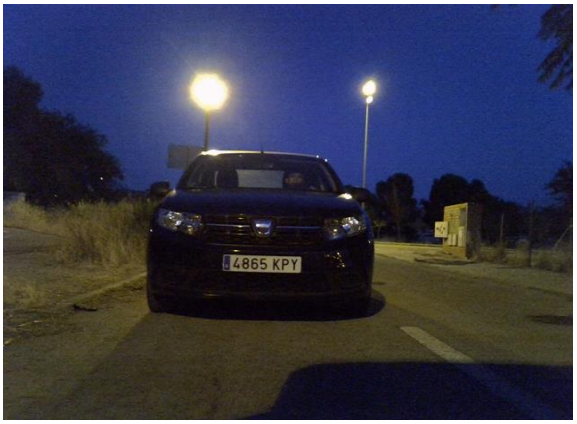

Figura 16: Captura con baja iluminación 1

Figura 17: Captura con baja iluminación 2

Figura 18: Captura con baja iluminación 3

Se puede observar que las imágenes son fácilmente legibles a simple vista, pero el programa es muy sensible a los reflejos o las sombras, lo cual provoca que a veces los confunda con matrículas o no recorte la matricula completa.

3.1.3 Recorte de la matrícula

Para poder obtener el texto de las imágenes primero habrá que localizar la ubicación aproximada de la matrícula , posteriormente se procesa esta y, por último, se extrae el texto siguiendo el flujo visto en la figura 19

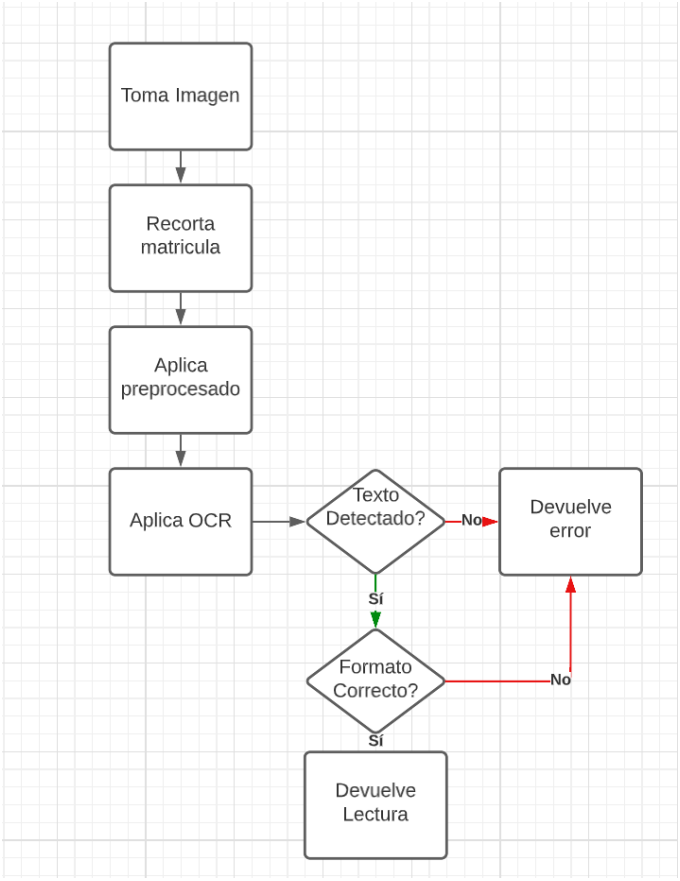

Figura 19: Detección matrícula

En el primer paso se va a detectar la matricula dentro de la imagen, para lo que se ha empleado un clasificador Haar.

Para poder procesar imágenes con OpenCV el primer paso será convertir una imagen, cargada previamente haciendo uso de la función “imread()”, a escala de grises, para esto se emplea la función cvtColor() disponible en la librería de OpenCV como se muestra en la Figura 20. Esta función se encarga de convertir una imagen de un espacio de color a otro, para trabajar con OpenCV tanto si se quiere la imagen en color como si quiere en blanco y negro, habrá que convertirla ya que el formato estándar es RGB, pero OpenCV hace uso de del formato BGR.

```
image = cv2.imread(location_input)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Figura 20: Cargar imagen y convertir a escala de grises

Para poder localizar la matricula se ha hecho uso de clasificadores Haar. Es un método eficaz de detección de objetos propuesto por Paul Viola y Michael Jones en el 2001.Es un enfoque basado en el aprendizaje automático en el que la función de la cascada se entrena a partir de muchas imágenes positivas y negativas para que luego se pueda usar para detectar objetos en otras imágenes.

Para entrenar el clasificador se requieren muchas imágenes positivas y muchas imágenes negativas, después hace falta extraer características en él, para eso se utilizan las siguientes características de Haar:

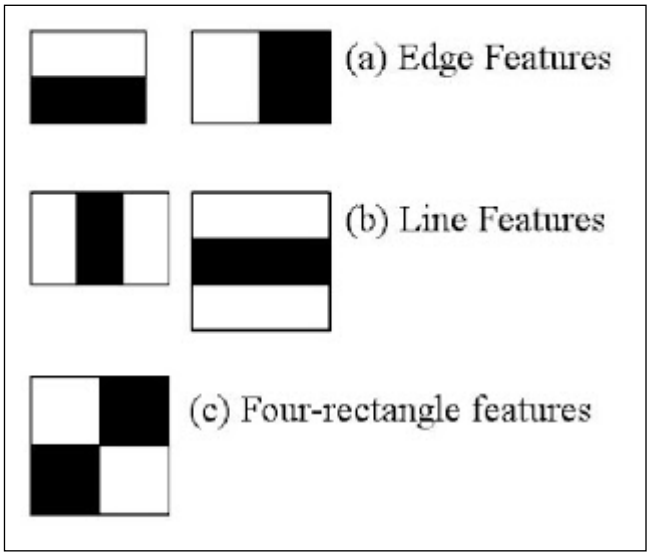

Ilustración 1: Haar features

Cada característica es un valor único obtenido al restar la suma de píxeles debajo del rectángulo blanco de la suma de píxeles debajo del rectángulo negro. Ahora todos los tamaños y ubicaciones posibles de cada kernel. Para cada cálculo de características hay que encontrar la suma de pixeles debajo de los rectángulos blanco y negro.

Una vez creado el clasificador hay que comenzar a entrenarlo. Partiendo de las características que se tienen, se pasa a calcular más y se va entrenando el clasificador hasta conseguir la tasa de aciertos deseada, dando mayor prioridad a unas características que a otras. En cada clasificación se aumenta el peso de las imágenes clasificadas, de forma que se va creando un clasificador cada vez mejor, de forma que el clasificador final es la suma ponderada de todos los clasificadores.

Por último, en la imagen se aplica el concepto de cascade, de modo que se van aplicando las características desde las de mayor peso a las de menor. Con esto se logra un algoritmo más optimizado ya que solo se sigue analizando en caso de que los primeros resultados devuelven resultados correctos, Se ha decidido hacer uso de un algoritmo que está ya entrenado para detectar matriculas y se ha logrado obtener una tasa de aciertos es muy buena, mucho mejor que detectando los contornos sin a ver hecho uso de del clasificador ya que se evitan los contornos tales como los faros del vehículo o cajas blancas.

En la Figura 21 se observa el proceso a seguir para cargar el clasificador desde un archivo XML. Este no ha sido creado por el alumno ya que sobrepasaría la capacidad de desarrollo para este proyecto,

de modo que se hace uso de uno ya entrenado previamente. Este se emplea para detectar las matrículas en la imagen haciendo uso de la función detectMultiScale() incluida en OpenCV y pasando como parámetros la imagen en la que se va a buscar los elementos, el factor de escala y el número de vecinos que debe tener para que se considere valido. En este proyecto, el factor de escala se ha establecido a 1.05, lo que significa que se irá reduciendo un 5% en cada ejecución y el número de vecinos a 7, lo que se refiere al número de vecinos que debe de tener cada rectángulo para ser considerado valido, a mayor valor se obtienen menos resultados, pero con mayor calidad de detección.

```
n_plate_detector = cv2.CascadeClassifier('haarcascades/haarcascade_plate_number.xml')
detections = n_plate_detector.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=7)
```

Figura 21: Clasificador en Cascada

Una vez obtenidos los resultados se añadirán recortes del ROI (Region Of Interest) de la imagen a un array de imágenes (Figura 22) para analizar solo las zonas que pueden resultar de interés, haciendo que se detecte mejor el contorno de la matricula. Ya que el algoritmo no devuelve las esquinas de la detección, sino que la posición una de ellas junto con el alto y el ancho, esto puede provocar que se recorte parte de la matricula en caso de que esta esté muy torcida, para evitar esto se ha aplicado un margen de error al recorte.

```
results = []
for (x, y, w, h) in detections:
    cv2.rectangle(image, (x - 10, y - 10), (x + w + 10, y + h + 10), (0, 255, 255), 2)
    cv2.putText(image, "Posible matricula detectada", (x - 50, y - 50),
                cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 255, 255), 2)
    results.append(image[y-10:y + h + 5, x - 5:x + w + 5])
```

Figura 22: Almacenar resultados de aplicar homografía

Al aplicar esto se obtendrán todas las detecciones que el clasificador consideré correctas, que en los casos más favorables serán un único resultado como en la figura 23, pero puede suceder que se obtengan detecciones incorrectas como se ve en la figura 24 ya que el clasificador no es perfecto y pude confundir reflejos u otras formas blancas rectangulares con matrículas. En este segundo caso (Figura 25) habrá que analizar las detecciones incorrectas para decidir en cuál de ellas esta presenta la matricula.

Figura 23: Una única matricula detectada

Figura 24:Matricula Recortada

Figura 25: Varias matriculas detectadas

Una vez recortada la imagen se va a corregir la perspectiva presente en las matrículas que puede aparecerán función de las diferentes alturas de los vehículos o el ángulo con el que se acercan. Para lograr esto el primer paso será encontrar los puntos de las esquinas de estas, una vez obtenidos se obtiene la matriz de la perspectiva pasando los puntos de las esquinas y los puntos en los que se desea que se ubique la matricula transformada. Por último, se aplica la transformación a la imagen para que las esquinas se ubiquen en esos puntos.

El primer paso será cargar la imagen recibida por parte del clasificador, que es con la que se va a trabajar, de modo que se parte de la imagen de la Figura 24:

Tras esto se aplicará un filtrado bilateral (Figura 26). Un filtrado a una imagen significa aplicar un suavizado a la imagen para eliminar ruido de esta, haciendo que los colores sean más uniformes y sea más fácil distinguir las formas. En OpenCV se pueden aplicar tres filtros distintos:

Filtro Gaussiano: como dice el nombre, se aplica el método de la campana de Gauss para aplicar un kernel que escanea cada píxel y utiliza un promedio ponderado para reemplazar el valor del píxel.

Filtro de Mediana: Este filtro recorre todos los pixeles y reemplaza su valor con la mediana de los pixeles vecinos

Filtro Bilateral: Este se encargará de reemplazar la intensidad de cada píxel con un promedio ponderado de los valores de intensidad de los píxeles cercanos. Este peso puede basarse en una distribución gaussiana. Los pesos se calculan teniendo en cuenta tanto la distancia entre los píxeles tanto como las diferencias entre estos.

En este proyecto el tipo de filtrado aplicado será el Bilateral, ya que este permitirá obtener una imagen con menos ruido en las partes que son de un mismo color, pero manteniendo los bordes facilitar la detección de contornos, el resultado de esto se observa en la figura 27.

```
    blur = cv2.bilateralFilter(img, 9, 10, 10)
```

Figura 26: Filtrado bilateral

Figura 27: Comparativa entre la imagen original (superior) y la suavizada (inferior)

Ahora se va a binarizar la imagen para tratar de obtener el ROI. Binarizar una imagen es el proceso mediante el cual se eliminar los colores dejando únicamente el blanco y el negro. Para esto se establece un umbral y, los valores por encima de ese umbral serán blancos y los que estén por debajo negros. Esto se hace para poder diferenciar mejor los contornos.

Para poder hacer esto hay que aplicar la función threshold de OpenCV como en la Figura 28.

Al aplicar binarizado, se debe escoger el algoritmo deseado, se ha decidido hacer uso de OTSU, que es un algoritmo que se encarga de seleccionar el threshold para la binarización de forma automática. Este es muy útil en caso de que el histograma tenga dos picos diferenciados, se ha decidido usar este ya que al venir las matrículas en blanco y negro se observan claramente los dos picos en torno a los que se realizara la diferenciación.

Una vez aplicado el proceso de binarización se obtiene una matrícula similar a la obtenida en la Figura 29.

```
thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
```

Figura 28: Binarizar imagen

Figura 29: Matrícula binarizada

Posteriormente se va a erosionar(Figura 30) la imagen para suavizar los bordes aplicando una máscara de 3x3, logrando delimitar mejor las zonas claras y oscuras. Esto provoca que se pierda mucha calidad en la imagen, de modo que únicamente se usará para localizar la matrícula y así obtener mejor los puntos de los contornos. En la Figura 31 se observa que se ha perdido mucho detalle en la imagen, pero la forma de la matricula sigue siendo perfectamente visible.

```
kernel = np.ones((3, 3), np.uint8)  # Mascara de unos 3x3
ero = cv2.erode(thresh, kernel, iterations=1)
```

Figura 30: Erosionar imagen

Figura 31: Matrícula erosionada

Una vez hecho esto se obtendrán los contornos, un contorno se puede definir como una serie de puntos unidos entre si con un mismo color o intensidad, para lograr obtener los contornos en Python hay que hacer uso de la función findContours (Figura 32), a esta función hay que pasarle tres parámetros, la imagen, el modo de detección (contornos internos, externos o completo) y el método empleado (Simple, si queremos que se obtengan solo los puntos más relevantes del contorno o None si queremos que devuelva todos los puntos que se han obtenido). Para poder aproximar a un polígono correctamente habrá que obtener todos los puntos del contorno e ir simplificando.

Primero se realizará una búsqueda para obtener todos los contornos presentes en la imagen y posteriormente se escogerá el contorno de mayor área. El resultado de aplicar estas funciones se puede observar en la figura 33.

```
contours, hierarchy = cv2.findContours(dil, cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_NONE)
cnts2 = sorted(contours, key=cv2.contourArea, reverse=True)[:1]
```

Figura 32: Obtener el mayor contorno

Figura 33: Matrícula con contorno

Con el contorno obtenido, se simplificará haciendo uso de la función approxPolyDP (Figura 34). Esta función se encarga de aproximar una curva o un polígono con un menor número de vértices, para esto lo que hará será unir un punto con un tercero y, si la distancia al segundo punto es inferior a la establecida, este punto intermedio se elimina. Para aproximar cada vez más se ha creado en un bucle que va aumentando la distancia permitida hasta que únicamente queden cuatro puntos, en ese momento se abandona el bucle y se comprueba que solo queden cuatro. Los puntos resultantes son los mostrados en la figura 35.

```
epsilon = 0.01 * cv2.arcLength(cnt, True)
approx = cv2.approxPolyDP(cnt, epsilon, True)
while len(approx) > 4:
    epsilon = epsilon + 0.1
    approx = cv2.approxPolyDP(approx, epsilon, True)
```

Figura 34: Obtener esquinas

Figura 35: Esquinas detectadas (puntos rojos)

Una vez obtenidas las cuatro, comprobando primero que haya un número exacto de cuatro posibles esquinas, se calcula el centro de masas para poder identificar en que posición se encuentra la primera esquina. Sabiendo la posición relativa de esta frente al centro y que el array de esquinas siempre se obtiene en el mismo sentido, se puede ordenar estas(Figura 36) haciendo que las esquinas siempre estén en la misma posición.

```
centro = np.mean(approx, axis=0)[0]
primero = approx[0, 0]
if primero[0] <= centro[0]:
    if primero[1] <= centro[1]:
        input = np.float32([[approx[0, 0, 0], approx[0, 0, 1]], [approx[3, 0, 0], approx[3, 0, 1]],
                            [approx[2, 0, 0], approx[2, 0, 1]], [approx[1, 0, 0], approx[1, 0, 1]]])
    else:
        input = np.float32([[approx[3, 0, 0], approx[3, 0, 1]], [approx[2, 0, 0], approx[2, 0, 1]],
                            [approx[1, 0, 0], approx[1, 0, 1]], [approx[0, 0, 0], approx[0, 0, 1]]])

else:
    if primero[1] <= centro[1]:
        input = np.float32([[approx[1, 0, 0], approx[1, 0, 1]], [approx[0, 0, 0], approx[0, 0, 1]],
                            [approx[3, 0, 0], approx[3, 0, 1]], [approx[2, 0, 0], approx[2, 0, 1]]])
    else:
        input = np.float32([[approx[2, 0, 0], approx[2, 0, 1]], [approx[1, 0, 0], approx[1, 0, 1]],
                            [approx[0, 0, 0], approx[0, 0, 1]], [approx[3, 0, 0], approx[3, 0, 1]]])
```

Figura 36: Ordenar esquinas

Teniendo ya las coordenadas de los puntos ordenadas, se calcula el ancho y el alto que tendrá la matricula transformada, la primera esquina se establecerá en el punto (0,0) y el reto se calcularán a partir de esta. La transformada tendrá el doble de ancho y de alto que la matricula existente en la imagen original. Teniendo las coordenadas se obtendrá la ubicación de la matricula con la perspectiva corregida, se hará uso de la función “getPerspectiveTransform()” (Figura 37), que nos calculará una matriz de transformadas de 3 x 3.

El proceso de creación de esta matriz de 3x3 se llama Homografía, una función se encargará de mapear todos los puno de una imagen a los que se corresponden en otra. Para que esta funcione correctamente se requieren al menos cuatro puntos en cada una de las imágenes, que se usaran para hacer coincidir los de una con los de la otra y así poder transformar la imagen.

Se le pasará esta matriz a la función “warpPerspective()” junto con la imagen filtrada y devolverá la imagen corregida. Las diferencias entre la imagen de entrada y la salida, ya corregida, se muestra en la figura 38.

```
ancho_AD = np.sqrt(((input[0, 0] - input[1, 0]) ** 2) + ((input[0, 1] - input[1, 1]) ** 2))
ancho_BC = np.sqrt(((input[3, 0] - input[2, 0]) ** 2) + ((input[3, 1] - input[2, 1]) ** 2))
ancho = max(int(ancho_AD), int(ancho_BC)) * 2

alto_AB = np.sqrt(((input[0, 0] - input[3, 0]) ** 2) + ((input[0, 1] - input[3, 1]) ** 2))
alto_CD = np.sqrt(((input[2, 0] - input[1, 0]) ** 2) + ((input[2, 1] - input[1, 1]) ** 2))
alto = max(int(alto_AB), int(alto_CD)) * 2

input_pts = np.float32([input[0], input[3], input[2], input[1]])
output_pts = np.float32([[0, 0],
                         [0, alto - 1],
                         [ancho - 1, alto - 1],
                         [ancho - 1, 0]])

M = cv2.getPerspectiveTransform(input_pts, output_pts)

img = cv2.warpPerspective(blur, M, (ancho, alto), flags=cv2.INTER_LINEAR)
```

Figura 37: Corregir perspectiva

Figura 38: Matrícula sin perspectiva

3.1.4 Obtener texto

Teniendo la matricula con la perspectiva corregida, se aplicará un último paso antes de reconocer el texto de esta. Se dilatará la imagen para que los caracteres sean más finos y se volverá a binarizar (Figura 39).

Esto se hace para que los caracteres sean más fácilmente reconocibles por parte de TesseractOCR, de este modo se obtiene una imagen más clara y legible como se observa en la figura 40.

```
kernel = np.ones((2, 2), np.uint8)  # Mascara de unos 2x2
img_dilation = cv2.dilate(img, kernel, iterations=3)

ret, thresh1 = cv2.threshold(img_dilation, 120, 255, cv2.THRESH_BINARY +
                             cv2.THRESH_OTSU)

cv2.imshow("P_resultBin", thresh1)
```

Figura 39: Dilatar y binarizar para mejorar la lectura

Figura 40:Comparativa entre matricula sin tratar y tratada

Para reconocer el texto de la imagen se hace uso del programa TesseractOCR, que en Python se llama Pytesseract (Figura 41), y que se ha configurado para que únicamente reconozca caracteres que pueden estar presentes en las matrículas, removiendo las vocales y estableciendo unos patrones posibles, que son los que se emplean en las matrículas de España, tanto las antiguas como nuevas.

```
texto = pytesseract.image_to_string(thresh1, lang='eng',
                                    config='--psm 13 --oem 3  -c tessedit_char_whitelist=0123456789BCDFGHJKLMNPQRSTVWXYZ')
```

Figura 41: obtención de texto

Como se puede ver se hace uso de la función image_to_string disponible en pytesseract a la que habrá que pasarle tres parámetros:

El primero hace referencia al archivo donde estará ubicada la imagen con el texto, del que obtendremos la matricula.

El segundo hace referencia al idioma, que se empelará para que únicamente reconozca caracteres del idioma inglés, que coinciden con los de España, a excepción de la ñ que no aparece en las matrículas.

El ultimo parámetro es el más importante, en este se configura la forma en la que se leerá el texto, estableciendo que tiene que buscar, este tendrá configuraciones posibles tales como psm, oem o whitelist.

Psm hace referencia a la segmentación que emplea, las posibles opciones son:

1. Orientation and script detection (OSD) only.
2. Automatic page segmentation with OSD.
3. Automatic page segmentation, but no OSD, or OCR. (Not implemented)
4. Fully automatic page segmentation, but no OSD. (Default)
5. Assume a single column of text of variable sizes.
6. Assume a single uniform block of vertically aligned text.
7. Assume a single uniform block of text.
8. Treat the image as a single text line.
9. Treat the image as a single word.
10. Treat the image as a single word in a circle.
11. Treat the image as a single character.
12. Sparse text. Find as much text as possible.
13. Sparse text with OSD.
14. Raw line. Treat the image as a single text line.

Oem hace referencia al motor que se usará para el reconocimiento de caracteres y se puede ajustar en 4 niveles:

1. Legacy engine only.
2. Neural nets LSTM engine only.
3. Legacy + LSTM engines.
4. Default, based on what is available.

El ultimo parámetro se refiere a la whitelist, que son los caracteres permitidos, esto hace que solamente se puedan reconocer estos, lo que hace que el reconocimiento sea más preciso ya que siempre intenta encontrar estos en el texto.

La llamada a esta función es lo único que hace falta para devolver una cadena de texto con el contenido de la imagen, así que ahora habrá que comprobar que se haya leído correctamente y que, en caso de que se hayan detectado caracteres de más, estos se excluyan.

Para lograr esto, se ha implementado un reconocimiento de patrones (Figura 42), este lo que hará será buscar subcadenas que correspondan con el formato de matrículas español haciendo uso de la función re. Este es un módulo que permite la identificación de expresiones regulares en cadenas de texto, las expresiones regulares son patrones que se interpretan como un conjunto de instrucciones. Posteriormente la entrada se comprueba para ver si coincide con estos patrones.

La función search() tratara de buscar el patrón dentro del texto y devolverá un objeto Match en caso de que lo encuentre. Si no encuentra coincidencias devuelve None.

Si se hubiera contado con una cámara de mayor resolución se podría incluso detectar el identificador del país en la matrícula y así hacerlo coincidir con el formato de los países. Esto se podría implementar como una mejora del proyecto simplemente modificando la cámara y haciendo que cuando se recorte esta se añada un 7% del ancho de esta a la izquierda.

```
mat = ""
m = re.search('[0-9][0-9][0-9][0-9][A-Z][A-Z][A-Z]', texto.replace(" ", ""))
if m is None:
    m = re.search('[A-Z][A-Z]?[0-9][0-9][0-9][0-9][A-Z][A-Z]?', texto.replace(" ", ""))
    if m is None:
        mat = "ERROR"
    else:
        mat = m.group(0)

else:
    mat = m.group(0)
```

Figura 42: Reconocimiento de patrones

Se ha empleado la clase re de Python para tratar de buscar subcadenas que coinciden con un determinado patrón. Se puede observar que hay dos comprobaciones, la primera de ella se encarga de tratar de localizar matriculas actuales, que están formadas por cuatro número y tres letras. En caso de que esta no se encuentre, se buscara otro patrón distinto que corresponde con las matrículas antiguas. Estas pueden estar formadas de varias formas distintas, empiezan por una o dos letras, dependiendo de la provincia en la que este matriculado, por ejemplo, M de Madrid o GU de Guadalajara, después tendremos cuatro números y al final una o dos letras, dependiendo del tamaño que tiene la provincia.

Si se logra encontrar una subcadena que corresponda con el patrón se devolverá la subcadena como matricula, si no la encuentra se devolverá el mensaje “ERROR”.

3.1.5 Comparar detecciones

El programa está configurado para tomar cinco imágenes y decidir cuál es la lectura correcta, en caso de que se devuelvan varias distintas. Esto se ha hecho para tratar de evitar falsas detecciones ya que, mientras se toman las fotos, el vehículo sigue avanzando, de modo que se captan imágenes desde distintos ángulos y a distintas distancias. Como se observó en la primera función, una vez realizadas las capturas se van guardando los resultados del escaneo en un array. Una vez obtenido el array de matrículas se llamará a la función "seleccionar_matricula()" (Figura 43).

Esta función irá comparando los elementos del array para ver si hay dos que coincidan haciendo uso del siguiente bucle:

```
i = 0
while (i < (num_fotos - 1)) & (encontrado != 1):
    j = 1
    if lecturas[i] != "ERROR":
        if lecturas[j] != "ERROR":
            while ((i + j) < (len(lecturas))) & (encontrado != 1):
                if lecturas[i] == lecturas[i + j]:
                    matricula_iden = lecturas[i]
                    encontrado = 1
                j += 1
    i += 1
```

Figura 43: Comparación de los elementos

Se puede ver que hay un bucle que itera sobre i y otro sobre j, de esta forma se compara el primer elemento con el segundo, con el tercero… Después el segundo con el tercero, con el cuarto…. Y así sucesivamente hasta que coincidan dos de ellos o se hayan agotado todos los elementos. En caso de que uno de los elementos sea ERROR se saltará esa comparación y pasamos al siguiente.

En caso de que coincidan dos, se establece encontrado a 1 (true) y se guarda la matricula identificada, haciendo que se salga del bucle. Si sucede esto el resto del código se omite y se devuelve la matricula. En caso contrario se devolverá la primera matricula que se haya encontrado en el array que sea distinta de ERROR.

3.1.6 Notificar acceso

Tras recibir la detección, se comienza a procesar la entrada siguiendo el siguiente flujo (Figura 44):

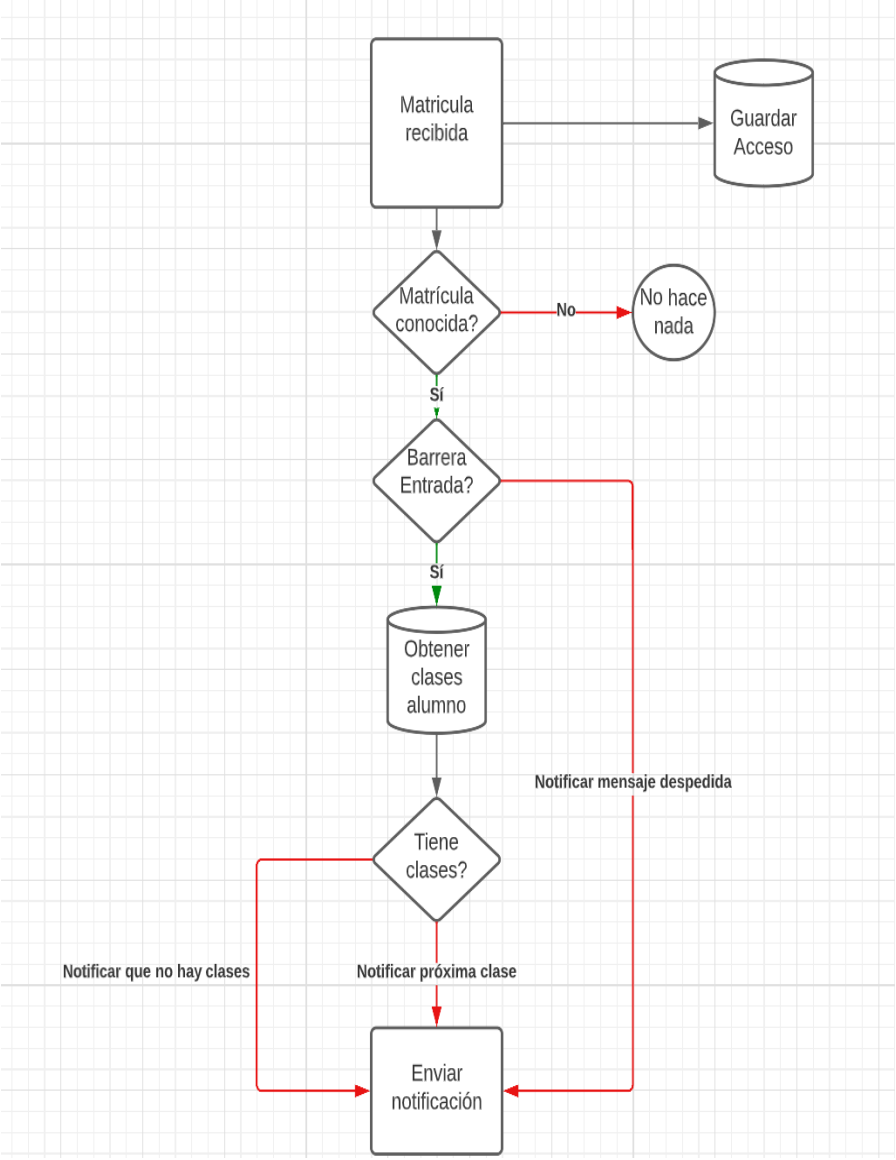

Figura 44: Recibir acceso

En caso de que la función del paso anterior haya sido capaz de reconocer una matrícula, esta será comprobada y, si todo es correcto, se notificara al servidor siguiendo los pasos vistos en la figura 45

```
if detectado != "ERROR":
    if detectado_anterior != detectado:
        detectado_anterior = detectado
        client.publish(topic, detectado)
        print("Matricula detectada: " + detectado)
    else:
        print("Este vehiculo ya ha pasado antes")
else:
    print("No se ha podido detectar")
```

Figura 45: Publicar acceso

Se puede observar que hace dos comprobaciones, en caso de que no se haya podido detectar no se hace nada, esto se deja así para que, en caso de que deseemos conectar la cámara a una barrera real, esta pueda transmitir una señal a esta para mostrar un mensaje o dar algún tipo de aviso para que el usuario acerque su tarjeta.

En caso de que si se haya detectado se comprobará que la matricula no coincida con la última entrada, para así evitar que si un coche se queda parado en frente de la barrera esta se levante varias veces o envié varios mensajes. En caso contrario, si es la primera vez que se detectase publica el mensaje en el servidor MQTT indicando el numero de la barrera y la matricula. Posteriormente se podrá añadir una señal para que se levante la barrera.

3.2 Bloque 2: Servidores y base de datos (página web)

En este bloque se va a incluir en el toda la parte de los servidores que se han montado y la base de datos.

Para poder cumplir con los objetivos de este proyecto, se ha creado base de datos en la que se guardan los datos de los alumnos, de los vehículos, de las clases… Por otra parte, se dispone de un servidor MQTT, otro encargado de guardar los servlets y procesar los datos del servidor MQTT, y otro que guardará la página HTML y PHP.

Se va a hacer uso de XAMPP para guardar los servidores. Este es un paquete de software libre que contiene diversos servidores de los que hemos hecho uso en este trabajo. Su nombre proviene de X (cualquier sistema operativo), Apache, MariaDB/MySql, PHP, Pearl.

Lo motivos por los que se decidió hacer uso de este debido a la comodidad de uso que supone ya que evita tener que instalar cada componente por separado, sino que es un completo paquete de software que incluye todo. Se ha hecho uso de MySql como gestor de la base de datos, de apache junto con PHP para guardar el servidor que se accede mediante la web y de Tomcat para hacer uso de los Servlets. En la Ilustración 3 se muestra el panel de control de XAMPP

3.2.1 Base de datos (MySQL)

Para la base de datos se ha elegido MySQL ya que está incluido en XAMPP y ya se había trabajado con este sistema anteriormente. Por otra parte, es también es la base de datos de código abierto más popular del mundo.

En primer lugar, se ha creado la base de datos “controlaccesos” y se ha configurado la base para que esta pueda ser accesible desde cualquier parte, de modo que se puedan conectar los dispositivos y se puedan cargar los datos.

Posteriormente se han creado las tablas, comenzando por usuario, que se encargara de guardar los datos de los alumnos:

La primera tabla será la encargada de guardar los datos personales de los alumnos (Figura 46), estos datos únicamente simulan una tabla real ya que, si se fuera a implementar esto en un entorno real, esta tabla vendría dada por el cliente y únicamente se haría uso de ella. La última entrada por ahora no se usa, se emplea para saber si un alumno sigue matriculado en la universidad, permitiendo así seguir guardando sus accesos y sus dato para consultas, aunque no se le permita el acceso al establecimiento.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**ID** 🔑	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	2	**NumMatricula**	int(11)			Sí	*NULL*			Cambiar Eliminar Más
☐	3	**Nombre**	varchar(20)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**Apellido**	varchar(20)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	5	**Telefono**	int(11)			Sí	*NULL*			Cambiar Eliminar Más
☐	6	**Email**	varchar(30)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	7	**Password**	varchar(256)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	8	**FechaCaducidad**	date			No	*Ninguna*			Cambiar Eliminar Más
☐	9	**Activo**	tinyint(1)			No	1			Cambiar Eliminar Más

Figura 46: Tabla Usuario

La tabla vehículo se encarga de guardar los datos del vehículo (Figura 47), guarda la matrícula, el propietario, si está activo actualmente y si se encuentra dentro de las instalaciones. Se guarda el estado del vehículo ya que un usuario puede modificar sus vehículos, y, de este modo, se sigue teniendo en cuenta en el historial de accesos una vez se ha eliminado.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**Matricula** 🔑	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	2	**Propietario** 🔑	varchar(20)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**Dentro**	tinyint(1)			No	*Ninguna*			Cambiar Eliminar Más
☐	4	**Activo**	tinyint(1)			No	1			Cambiar Eliminar Más

Figura 47: Tabla vehículo

La tabla barrera guarda los datos de las múltiples barreras existentes (Figura 48), cada barrera tiene un código identificador único y un booleano que indica si es de entrada o de salida, estos datos se usan para guardar los accesos, los otros dos datos son informativos, uno de ellos es la dirección, que se refiere al nombre de la vía en la que está ubicado y la organización a la que pertenece la barrera.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**id**	int(11)			No	*Ninguna*			Cambiar Eliminar Más
☐	2	**direccion**	text	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**organizacion**	varchar(30)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**entrada**	tinyint(1)			No	*Ninguna*			Cambiar Eliminar Más

Figura 48: Tabla Barrera

La tabla accesos guardará el historial de accesos (Figura 49) de todos los vehículos que han pasado por alunas barreras, se guarda la barrera por la que se ha pasado, la fecha en la que se produjo el acceso y el vehículo que ha pasado.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**Barrera**	int(11)			No	*Ninguna*			Cambiar Eliminar Más
☐	2	**Fecha**	date			No	current_timestamp()			Cambiar Eliminar Más
☐	3	**Vehiculo**	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 49: Tabla Acceso

La tabla asignaturas guarda los datos de las asignaturas existentes (Figura 50) para que los alumnos se puedan apuntar a ellas. Esta tabla al igual que la de los usuarios no es aplicable en un entorno real ya que estos datos están guardados en las bases propias de la empresa. Se guarda el código de la asignatura, el nombre de esta y los datos del grupo y el subgrupo que puede tener.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**codigo** 🔑	int(11)			No	*Ninguna*			Cambiar Eliminar Más
☐	2	**grupo** 🔑	varchar(8)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**nombre**	varchar(30)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**subgrupo**	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 50: Tabla Asignatura

En esta tabla Horario se guarda el horario de las asignaturas que existen(Figura 51), para esto se debe almacenar el aula, el día, la hora de inicio y de final de cuando se imparte esta asignatura, también tenemos que guardar el código de la asignatura y el grupo de esta, por ejemplo, una combinación de esto sería Programación Mañana o Matemáticas Tarde. Esta tabla no es aplicable en un entorno real ya que estos datos están guardados en las bases propias de la organización.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**aula**	varchar(5)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	2	**codigo** 🔑	int(11)			No	*Ninguna*			Cambiar Eliminar Más
☐	3	**grupo** 🔑	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**dia**	int(1)			No	*Ninguna*			Cambiar Eliminar Más
☐	5	**inicio**	time			No	*Ninguna*			Cambiar Eliminar Más
☐	6	**fin**	time			No	*Ninguna*			Cambiar Eliminar Más

Figura 51: Tabla Horario

En la tabla AsignaturasElegidas se guardarán las asignaturas en las que se ha matriculado cada alumno (Figura 52) para poder enviarle las notificaciones y mostrar el horario únicamente de las asignaturas en las que se ha matriculado. Esta tabla no es aplicable en un entorno real ya que estos datos están guardados en las bases propias de la empresa.

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**codigo_asignatura**	int(10)			No	*Ninguna*			Cambiar Eliminar Más
☐	2	**grupo_asignatura**	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**ID_Alumno**	varchar(10)	utf8mb4_general_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 52: Tabla AsignaturasElegidas

Relaciones:

Usuario y vehículo tienen una relación 1 a n, un usuario puede tener múltiples vehículos, pero un vehículo únicamente puede pertenecer a un usuario.

Usuario y AsignaturasElegidas tiene una relación muchos a muchos, un alumno puede tener varias asignaturas elegidas y una asignatura puede tener a varios alumnos

Vehículo y accesos tiene una relación 1 a n, un vehículo puede tener múltiples accesos y cada acceso pertenece a un único vehículo.

Barrera y acceso tienen una relación 1 a n, una barrera puede tener múltiples accesos y cada acceso se corresponde con una única barrera.

Asignatura y horario tienen una relación muchos a muchos, una asignatura se puede dar en horas distintas y puede haber varias asignaturas en un mismo momento.

Asignatura y AsignaturasElegidas tienen una relación muchos a muchos, una asignatura puede ser elegida por múltiples alumnos y un alumno puede elegir varias asignaturas.

3.2.2 Server MQTT (Mosquitto)

Figura 2: Logo Mosquitto

MQTT (MQ Telemetry Transport) es el proceso más usado para enviar mensajes cuando hablamos de IOT. Es un protocolo de comunicación M2M (Machine to Machine) del tipo message queue que emplea el protocolo TCP/IP. Este se basa en el uso de tópicos en los que se publican mensajes, que recibirán todos los dispositivos que se hayan suscrito a ellos, conectándose al dispositivo central, llamado broker.

El funcionamiento de este es el siguiente, una vez iniciado el servicio los clientes envían mensajes CONNECT que lleva las credenciales de conexión y el bróker responde con un mensaje CONNACK, en el que informa si la conexión re ha realizado correctamente. Una vez conectado al bróker el cliente se suscribe a los tópicos empleando SUBSCRIBE para recibir los mensajes que se envíen a tales tópicos. Si lo que quiere es enviar un mensaje hará uso de PUBLISH, en este se incluye el mensaje y el tópico, de modo que todos los clientes que estén suscritos a ese tópico lo recibirán.

La calidad del envió de los mensajes viene dado por el QOS (Quality Of Service, que tiene tres configuraciones posibles. La más básica es 0, en este el mensaje solo se envía una vez, de modo que puede no recibirse y no se comprobará, después tenemos el 1, en el que se garantiza la entrega del menaje, de modo que se comprueba que el mensaje se haya recibido antes de dejar de enviarlo, en caso de que falle notificar la recepción puede entregarse el mensaje más de una vez. Por último, tenemos el nivel 2, en el que se asegura que el mensaje se entrega una única vez. Por otra parte, los casos en los que nos podemos decantar por emplear una opción menos segura, es porque los mensajes se reciben más rápidamente.

En cuanto a la seguridad del sistema, se hace uso del transporte SSL/TLS y la posibilidad de autentificación por usuario y contraseña o mediante certificado, generalmente se emplea una simple conexión por usuario y contraseña debido a las restricciones de potencia de los dispositivos de IoT. Se ha elegido este protocolo para él envió de notificaciones debido a su ligereza, lo que implica un bajo consumo de recursos, de modo que la aplicación móvil consuma lo mínimo posible.

Para la comunicación se hace uso del protocolo TCP/IP, siempre teniendo en cuenta los recursos limitados y tratando de minimizar el consumo.

El funcionamiento de Mosquitto es bastante sencillo de comprender, existe un bróker que es el servidor encargado de manejar la conexión, este se encarga de recibir y enviar los mensajes. También tenemos “Publisher” y “Subscribers”, los primeros son los que envían los mensajes a un determinado tópico y los segundos son los que reciben los mensajes, estos se tienen que suscribir a todos los tópicos de los que quieren recibir mensajes.

La instalación de este programa es bastante sencilla ya que únicamente hemos hecho uso del instalador para Windows y, posteriormente, se ha configurado el servidor para que acepte conexiones desde cualquier dirección IP y no requiera autenticación (Figura 53).

```
# ================================================================
# Listeners
# ================================================================
listener 1883 0.0.0.0
allow_anonymous true
```

Figura 53: Configuración Mosquitto

Una vez creado el servidor se ha configurado un bróker para que gestione las conexiones (Figura 54) que se encargará de manejar los mensajes que envían las barreras. Este primer servidor (montado en el servlet de Tomcat) se suscribirá todas las barreras existentes haciendo uso del siguiente bróker:

```
public class MQTTBroker
{
    private static int qos = 2;
    private static String broker = Datos.urlMQTT;
    private static String clientId = String.valueOf(System.nanoTime());
```

Figura 54: MQTT Broker

En este bróker se establecerá el QOS (Quality Of Service) a 2, lo que significa que cada notificación se enviará una única vez, asegurándose de que esta sea recibida por el destinatario. También se fijará la IP del servidor para que sea la del servidor de Mosquitto, en nuestro caso será la IP pública de nuestro ordenador y la id del cliente como el valor de la hora, para que cada vez que se suscriba use una diferente.

Por otra parte, crearemos el suscriber que recibirá los mensajes de la barrera, paro esto se ha creado una función que se conectara a la base de datos para obtener todas las barreras existentes y subscribirse a cada una de ellas y así poder recibir notificaciones de todas. Después se ha modificado la función encargada de recibir mensajes para que procese las notificaciones empleando la función VehicleArrived de la clase Logic, esta función sigue una serie de pasos:

Guarda el acceso en la base de datos.

Consulta si la barrera es de entrada o salida.

Si es de entrada se notifica al usuario su próxima clase, si es de salida se envía un mensaje de despedida, para esto se hace uso del Publisher.

3.2.3 Servlets y Backend (Tomcat)

Son programas java que se ejecutan en el servidor y se usa para construir páginas webs dinámicas, basadas en fuentes variables o que extraen datos de bases de datos. El funcionamiento de este es sencillo, recibe una petición del tipo GET o POST, la procesa haciendo uso del código Java y devuelve una respuesta al servidor que la ha invocado, en este proyecto se van a devolver respuestas en formato JSON. Estos servlets requieren un servidor web, se ha decidido hacer uso de Apache Tomcat ya que ofrece la capacidad de contener tanto las páginas HTML como los servlets y posee ventajas tales como ser "Open-source", de modo que es totalmente gratis, ser ligero, de modo que en un ordenador que o tenga demasiados recursos funcionará bien. También es un servidor independiente, lo cual permite que la aplicación siga funcionando sin depender del estado de la página web.

Para la aplicación se ha creado una serie de clases que se usarán para procesar los datos de la aplicación. Para esto hemos se ha creado una aplicación que se divide en una serie de paquetes:

datosConexiones: Paquete que únicamente tiene una clase en la que se guardan las direcciones IP para las conexiones.

db: En este paquete se guarda una clase que tiene las funciones que se usan para conectarse a la base de datos y las clases que se corresponden con las tablas de la base.

logic: En este paquete se guardan las funciones que procesan los datos de la aplicación junto con la clase que guarda el log y la que inicia el servicio MQTT.

mqtt: Paquete que gestiona las conexiones MQTT.

servlets: Esta clase guarda las funciones que se podrán llamar desde la página web.

Ahora vamos a ver cada paquete en parte para ver que clases tenemos en cada uno y cuál es la funcionalidad de estas:

El paquete datosConexiones (Figura 55) guardara dos valores estáticos, que tendrán las URL que se usan para conectarse a la base de datos y al servidor de Mosquitto.

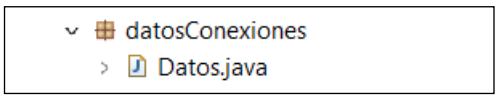

Figura 55: Paquete Datos Conexiones

```
public static String urlDb = "jdbc:mysql://localhost:3306/controlaccesos";
public static String urlMQTT = "tcp://localhost:1883";
```

Este paquete guarda tres clases (Accesos, Usuarios y vehículo) que se corresponden con tablas de la base de datos y se usan para guardar los datos que se tratarán en la aplicación web (Figura 56), en cada clase dispondremos de los valores de las columnas existentes en la base de datos, también existirá un constructor que establece los valores por defecto que tendrán los elementos

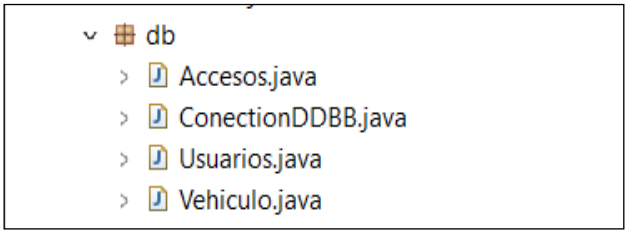

Figura 56: Paquete db

La otra clase, ConectionDDBB se usa para establecer la conexión y guardar los query a realizar:

Se hace uso del controlador MySQL jdbc Driver para poder establecer las conexiones con la base de datos, vamos a ver las principales funciones de esta clase:

obtainConnection: Establece la conexión con la base de datos (Figura 57) , para esto requiere del controlador de la base a utilizar y las credenciales de acceso:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
String sURL = Datos.urlDb;
conn = DriverManager.getConnection(sURL,"admin","passwd");
Log.logdb.debug("Se ha conectado a la base de datos.");
```

Figura 57: Conexión con la base

closeConnection: Cierra la conexión con la base de datos(Figura 58), de modo que se desconecta de esta.

```
Calendar calendar = Calendar.getInstance();
java.sql.Date date = new java.sql.Date(calendar.getTime().getTime());
Log.logdb.debug("Connection closed. Bd connection identifier: {} obtained in {}", con.toString(), date.toString());
con.close();
```

Figura 58: Cerrar conexión con la base de datos

Consultas a la base de datos, las query se devuelven a través de los elementos de tipo PreparedStatement:

```
public static PreparedStatement ExisteUsuario(Connection con)
  {
        return getStatement(con,"SELECT * FROM `usuario` WHERE `Email` = ? AND `Password` = ?;");
  }
public static PreparedStatement GetBarriers(Connection con)
  {
        return getStatement(con,"SELECT * FROM barrera");
  }
public static PreparedStatement GetMyVehicles(Connection con)
  {
        return getStatement(con,"SELECT * FROM `vehiculo` WHERE `Propietario` = ?;");
        }
public static PreparedStatement InsertAccess(Connection con)
  {
        return getStatement(con,"INSERT INTO `acceso` (`Vehiculo`, `Barrera`) VALUES (?,?) ;");
  }
public static PreparedStatement NewCar(Connection con)
  {
        return getStatement(con,"INSERT INTO `vehiculo`(`Propietario`, `Matricula`, `Dentro`) VALUES
        (?,?,0);"); }
public static PreparedStatement ExistCar(Connection con)
  {
        return getStatement(con,"SELECT * FROM `vehiculo` WHERE `Matricula` = ?;");
  }
public static PreparedStatement ActivateCar(Connection con)
  {
        return getStatement(con,"UPDATE `vehiculo` SET `activo`=1 WHERE `Propietario` = ? and `Matricula`
        = ?;");
  }
public static PreparedStatement GetAccesos(Connection con)
  {
        return getStatement(con,"SELECT * FROM `acceso`  JOIN Vehiculo on acceso.Vehiculo =
        Vehiculo.Matricula WHERE Vehiculo.Propietario = ? ");
  }
```

```
public static PreparedStatement GetDatosUsuario(Connection con)
   {
         return getStatement(con,"SELECT * FROM `usuario` WHERE id =  ? ");
   }
public static PreparedStatement ExistDNI(Connection con)
   {
         return getStatement(con,"SELECT * FROM `usuario` WHERE `ID`= ?;");
   }
public static PreparedStatement ExistMatriculaAlumno(Connection con)
   {
         return getStatement(con,"SELECT * FROM `usuario` WHERE `NumMatricula`= ?;");
   }
public static PreparedStatement ExistCorreo(Connection con)
   {
         return getStatement(con,"SELECT * FROM `usuario` WHERE `Email`= ?;");
   }
public static PreparedStatement RegistrarUsuario(Connection con)
   {
         return getStatement(con,"INSERT INTO `usuario`(`ID`, `Nombre`, `Apellido`, `NumMatricula`,
         `Telefono`, `Email`, `Password`, `FechaCaducidad`, `Activo`) VALUES (?,?,?,?,?,?,?,?,?) ;");
   }
public static PreparedStatement GetTipoBarrera(Connection con)
   {
         return getStatement(con,"SELECT entrada FROM barrera WHERE id= ?;");
   }
public static PreparedStatement GetLastAccesOfVehicle(Connection con)
   {
         return getStatement(con,"SELECT Fecha FROM `acceso` WHERE `Vehiculo` = ? order by Fecha limit
         1;");
   }
```

```
public static PreparedStatement GetProximaClase(Connection con) {

        return getStatement(con,"select * from asignatura left join horario on asignatura.codigo =
        horario.codigo "

                        + "WHERE asignatura.grupo = (SELECT grupo_asignatura FROM
        `asignaturaselegidas` WHERE ID_Alumno=(select propietario from vehiculo where
        matricula = ?)) "

                        + "and asignatura.codigo = (SELECT codigo_asignatura FROM
        `asignaturaselegidas` WHERE ID_Alumno=(select propietario from vehiculo where
        matricula = ?)) "

                        + "and horario.dia = (WEEKDAY(NOW())+1) "

                        + "and horario.fin >= now() "

                        + "order by horario.inicio;");

  }
```

El paquete logic (Figura 59) está formado por tres clases, la primera de ella es la que se encarga de guardar los loggers, a los que se les invocará para guardar los datos del log.

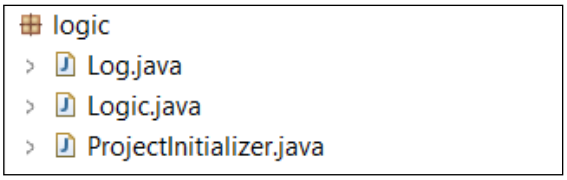

Figura 59: Paquete Logic

El segundo paquete es la clase Logic, esta tiene todas las funciones que se encargarán de tratar datos dentro de esta aplicación, las funciones que se han creado serán:

tryLogin: Esta función se encargará de intentar iniciar sesión recibiendo el email y la contraseña de un usuario, para lograr esto primero se crea la variable Usuario, que será donde se guardarán los datos, después se conectará a la base de datos e intentará obtener los datos de una entrada que contenga ese correo y esa contraseña, en caso de que se encuentre el usuario se devuelven sus datos, si no se devolverá el usuario creado por defecto.

getAccesos: Esta función se encarga de obtener todos los accesos de los vehículos pertenecientes a un usuario, para esto se crea un ArrayList de accesos que obtiene los datos devueltos por la consulta GetAccesos.

getMyVehicles: Esta función realizara dos consultas en la base de datos ya que, no solo devuelve los vehículos asignados a un propietario, sino que también el ultimo acceso de cada uno de ellos. Para esto en vez de devolver un array de elementos de la clase vehículo, devolverá uno de la clase accesos, de modo que se puede aprovechar esa clase sin tener que crear otra. Primero se obtendrán todos los vehículos cuyo propietario sea el usuario y, para cada uno de los elementos obtenidos, se consultará su ultimo acceso.

getDatosUsuario: Esta función devuelve los datos de un usuario en concreto mediante su DNI, esta se usa para poder mostrar los datos del alumno dentro de la página una vez se ha iniciado la sesión.

vehicleArrived: Esta función es llamada por el suscriber cuando se recibe un mensaje, primero guarda el acceso en la base de datos, después consulta si la barrera es de entrada o salida y, por último, si es de entrada se notifica al usuario su próxima clase, si es de salida se envía un mensaje de despedida. Para esto se hace uso del Publisher.

tryRegister: Esta función trata de añadir un usuario nuevo a la base de datos. Una vez se reciben los valores introducidos en el formulario, se comienza comprobando que el email tenga el formato correcto haciendo uso de Pattern Matcher, de igual forma se comprueba el formato del DNI y del teléfono. Si todo es correcto se comprueba que el DNI, la matricula (número de matrícula universitaria) y el correo no se encuentren en la base de datos y, si no se encuentra nada en la base de datos, se crea el nuevo usuario.

newCar: Función encargada de crear un nuevo vehículo y asignarlo a un usuario, esta comienza comprobado que este no se encuentre ya en la base de datos, em caso de que este se encuentre y este inactivo (el usuario lo ha borrado), se vuelve a activar. En caso de que no se encuentre se añade el vehículo nuevo y se le asigna a un usuario.

Por último, disponemos la clase **ProjectInitializer** que se llama al lanzar el servlet y se encarga de iniciar la conexión a Mosquitto.

En el paquete servlets (Figura 60) todas las clases que se podrán invocar desde la página web, todos estos reciben unos parámetros haciendo uso del método GET y devuelven un resultado en formato JSON.

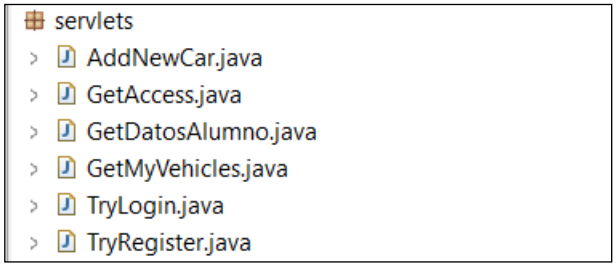

Figura 60: Paquete servlets

La estructura de todos los servlets es similar, así que tomaremos esta captura de la clase **AddNewCar** (Figura 61) como ejemplo.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    Log.log.info("-- Add new car to the DB--");
    //Preparamos las variables
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    String matricula=request.getParameter("matricula");
    String id=request.getParameter("id");
    try
    {
        //Llamamos a la funcion de logic
        String res =Logic.newCar(matricula,id);
        //Creamos la respuesta
        String jsonLogin = new Gson().toJson(res);

        Log.logjson.info("JSON Values=> {}", jsonLogin);
        //Devolvemos el json
        out.println(jsonLogin);
    } catch (NumberFormatException nfe)
    {
        out.println("-1");
        Log.log.error("Number Format Exception: {}", nfe);
    } catch (IndexOutOfBoundsException iobe)
    {
        out.println("-1");
        Log.log.error("Index out of bounds Exception: {}", iobe);
    } catch (Exception e)
    {
        out.println("-1");
        Log.log.error("Exception: {}", e);
    } finally
    {
        out.close();
    }
}
```

Figura 61: Estructura de un servlet

Se puede ver que se comienza estableciendo el tipo de texto de la respuesta y creando el printWriter, que será el documento JSON que devolveremos.

Ahora hay que obtener los valores (Strings) que se han pasado como parámetros, para eso se utiliza la función. getParameter de response, que recibe el nombre del parámetro deseado.

Posteriormente se empieza a trabajar con los datos, se observa la respuesta se obtiene llamando a una función de la clase Logic, a la que debemos convertir en JSON usando Gson().toJson(valores).

Por último, se guardarán los valores en la respuesta y se cierra la salida, lo que hará que se devuelva el contenido de esta a quien la haya invocado.

NOTA: Todas las funciones y las clases que se han creado cuentan con documentación propia y comentarios dentro del código.

3.2.4 Página web (Apache)

Para el servidor web se ha decidido hacer uso de páginas PHP con contenido en HTML y JavaScript.

La página principal de este servidor será **index.php** (Figura 62), que estará formada por la barra superior con las opciones de navegación y un container, que mostrará el resto de las ventanas, de modo que siempre nos encontraremos dentro del index.

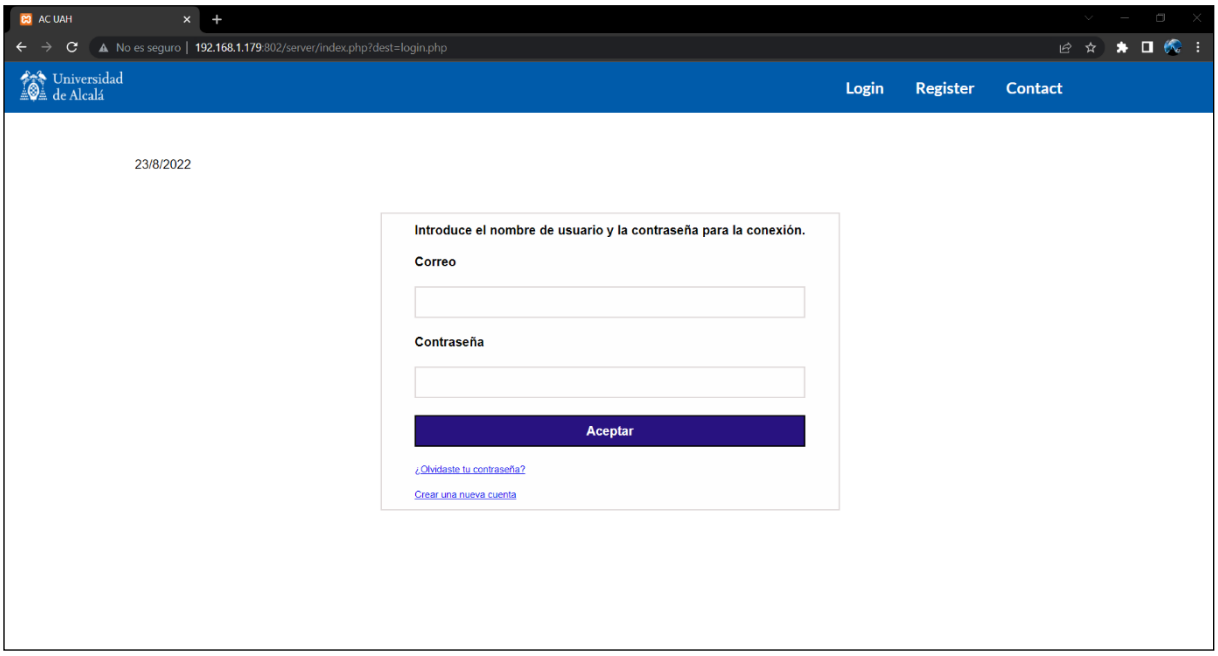

Figura 62: Página login

Vista usuario:

Por defecto se mostrará la página del login, que comenzará comprobando que la sesión no tenga un nombre de usuario asignado, es decir, que no se haya iniciado sesión previamente. En caso de que exista se redirigirá automáticamente a datos.php. Se puede ver que la pagina tiene un diseño simple, basado en la existente de la universidad de Alcalá. Mostrando dos entradas de texto para introducir el usuario y la contraseña, el botón para iniciar la sesión y otro dos botones, uno para registrarse y otro que se usara en caso de que se haya olvidado la contraseña.

Funciones:

Se hace uso de las sesiones php para guardar datos del usuario, para ello se llama a sesion_start() al principio del código. La función sesion_start() inicia la sesión en el navegador y se usa para poder guardar valores dentro de este, permitiendo que se acceda a estos posteriormente.

Para verificar que un usuario haya iniciado sesión se comprueba si la variable $_SESSION['nombredelusuario'] tiene algún valor, en caso de que la tenga significa que ya hay una sesión iniciada y se redirige a la pantalla de los datos, en caso contrario se queda esperando a que el usuario introduzca sus datos e inicie sesión.

Una vez se pulsa el botón Aceptar se conecta a la base de datos y comprueba que exista un usuario con ese login y esa contraseña. Si es así se inicia la sesión y se redirige a la pestaña con los datos.

Si se desea iniciar la sesión se comprobarán el usuario y la contraseña en la base de datos. En caso de que se encuentre se pasara a ver la ventana mostrando el historial de acceso y, en caso contrario, se mostrara una alerta (Figura 63) indicando que el usuario deseado no existe.

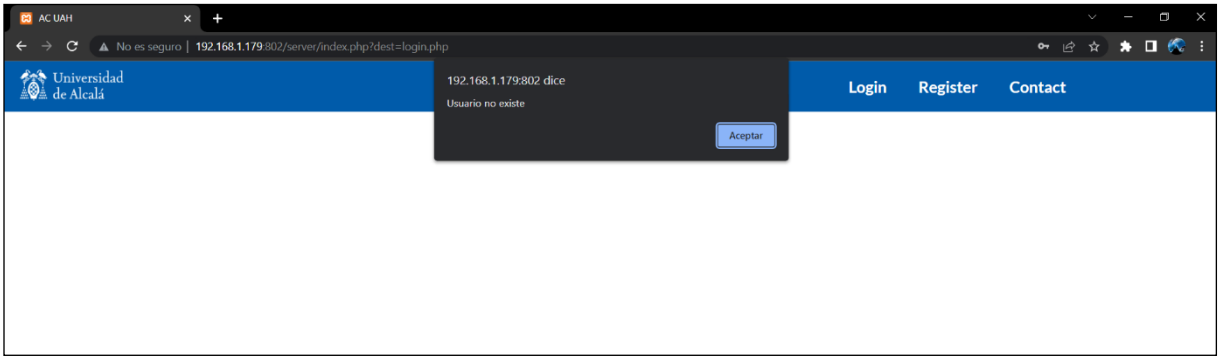

Figura 63: Error al iniciar sesión

El flujo seguido en el proceso de inicio de sesión será el mostrado en la Figura 64.

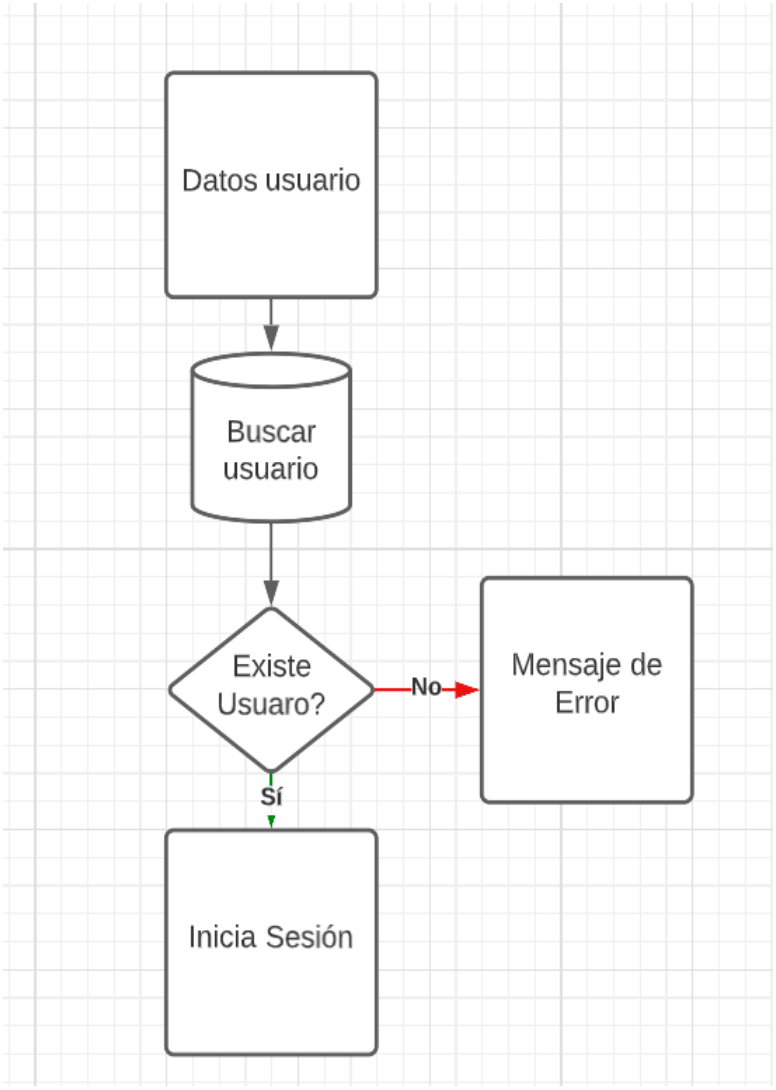

Figura 64: Iniciar Sesión

En caso de que se seleccione el botón de ¿Olvidaste tu contraseña?, se redirigirá a la siguiente página (Figura 65) que únicamente presentará una serie de formas en la que nos será posible recuperar la contraseña. Una posible mejora seria que el sistema pueda enviar un correo al usuario donde se le permita cambiar la contraseña o se le ofrezca una temporal que permita el acceso.

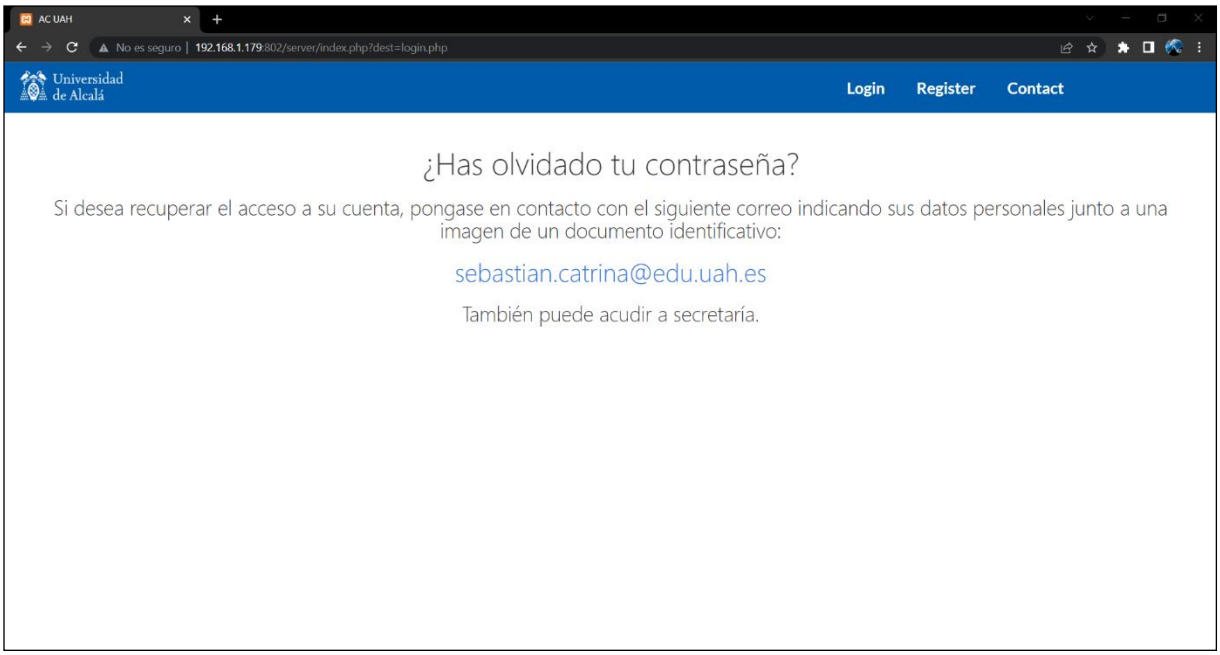

Figura 65: Contraseña olvidada

Al pulsar el botón de Registrarse se mostrará una ventana en la que se lo solicitarán los datos personales al alumno (Figura 66) .

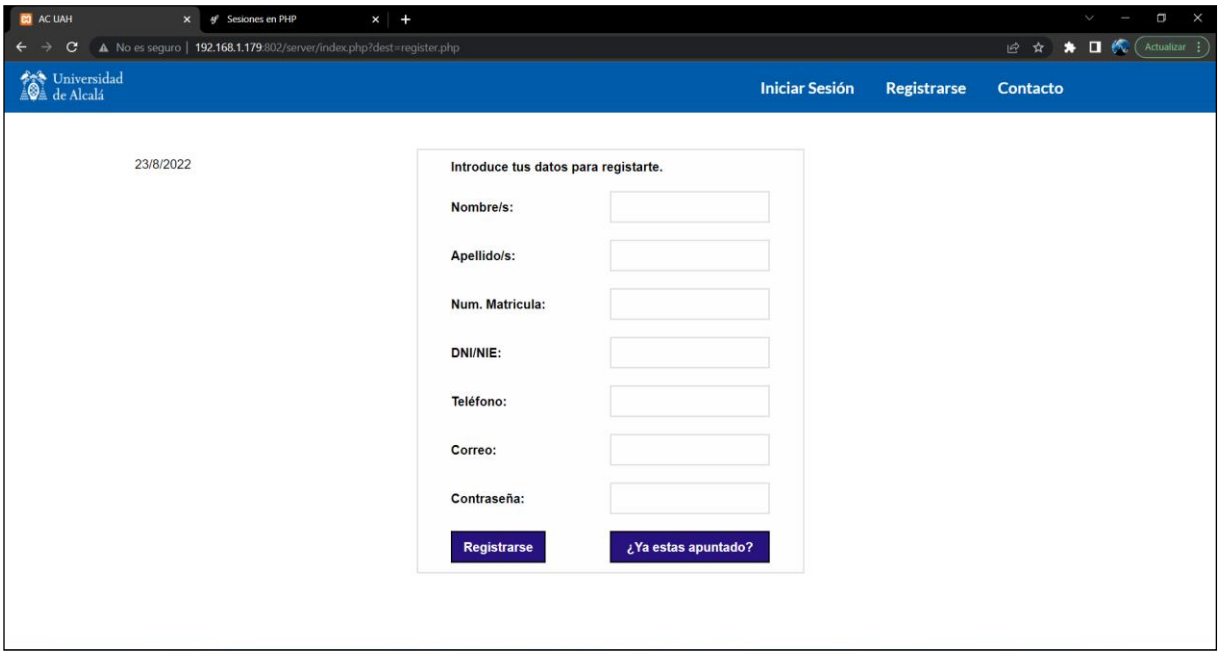

Figura 66: Pagina Registro

Vista cliente

Esta página presenta una interfaz sencilla en la que se solicitan los datos al alumno para que se pueda registrar. La función de esta es únicamente poder realizar pruebas y observar el funcionamiento del sistema ya que en un entorno real se hará uso de la página del cliente, concretando los datos que se piden.

Funciones

Se dispone un formulario donde estarán todas las entradas de texto, algunas de estas tienen un cierto tipo preestablecido para que comprueben que el formato de los datos introducidos es correcto.

Se ha añadido un EventListener al botón Registrarse que llama a la función que se encarga de registrar a un usuario, esta llamará al servlet TryRegyster y mostrará por pantalla el resultado de esta.

El flujo seguido durante el proceso de registro será el mostrado en la figura 67.

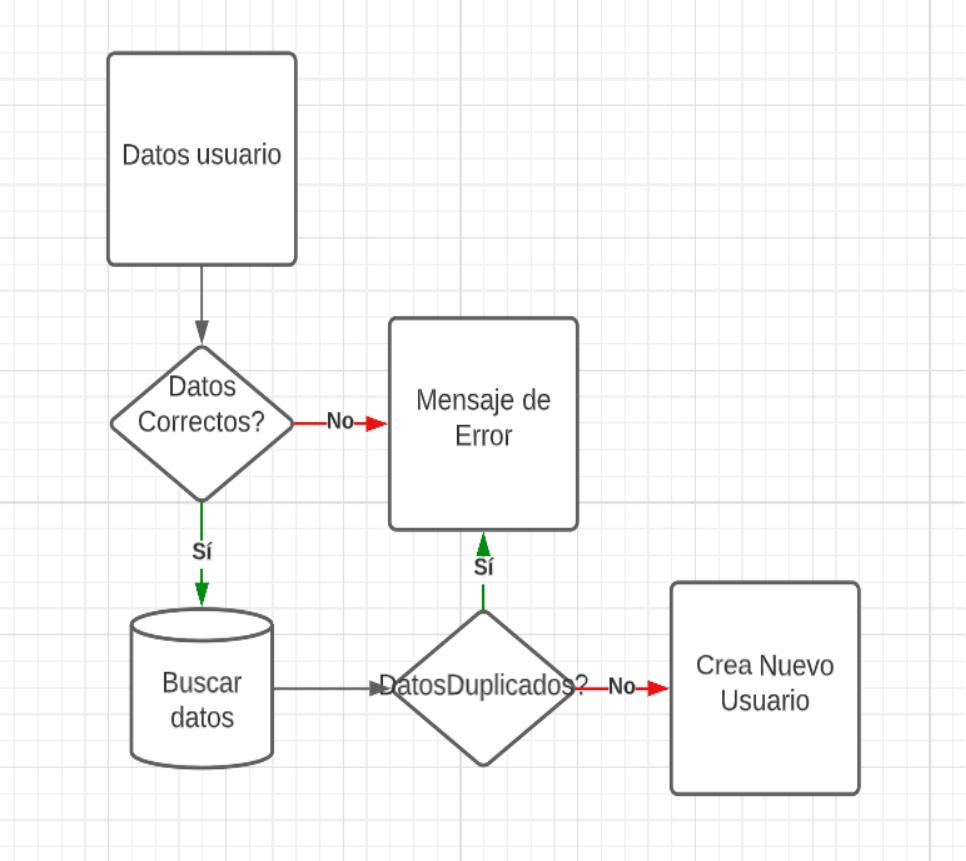

Figura 67: Registrar usuario

Dentro de la página existen algunas funciones que únicamente están disponibles desde la interfaz web, que son diferentes a las que se podrán usar desde la aplicación móvil.

Al iniciar sesión se muestra la ventana principal de la aplicación (Figura 68)

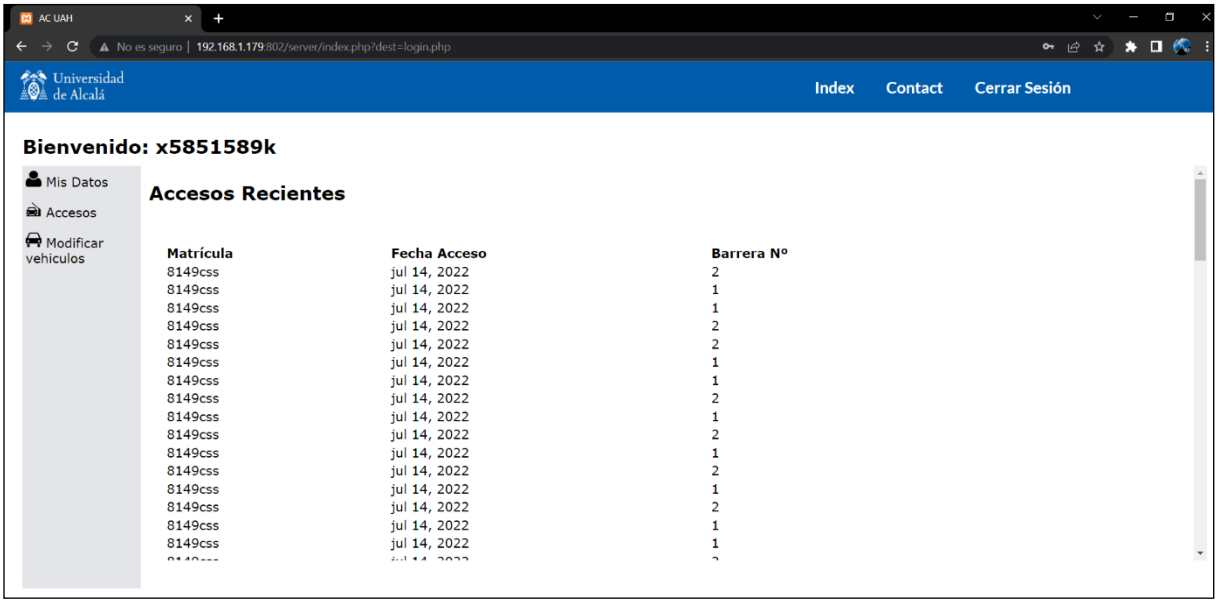

Figura 68: Vista al iniciar sesión

Vista cliente

Una vez se inicia la sesión se muestra un mensaje de bienvenida, una barra lateral que muestra las opciones posibles y el historial de accesos de los vehículos. Se puede ver que se muestra una tabla que indica la matrícula del vehículo que ha accedido, la fecha en la que lo hizo y por cuál de las barreras ha pasado.

Funciones

Esta vista contendrá una tabla con tres columnas, cuando se muestra eta vista se llama a la función cargarDatosUsr(), esta llamará al Servlet GetAccess, que devolverá un JSON con todos los accesos de los vehículos de un determinado usuario, con estos datos se llenará la tabla.

Esta pantalla mostrar los datos personales del alumno junto con una lista de los vehículos con su ultimo acceso(Figura 69)

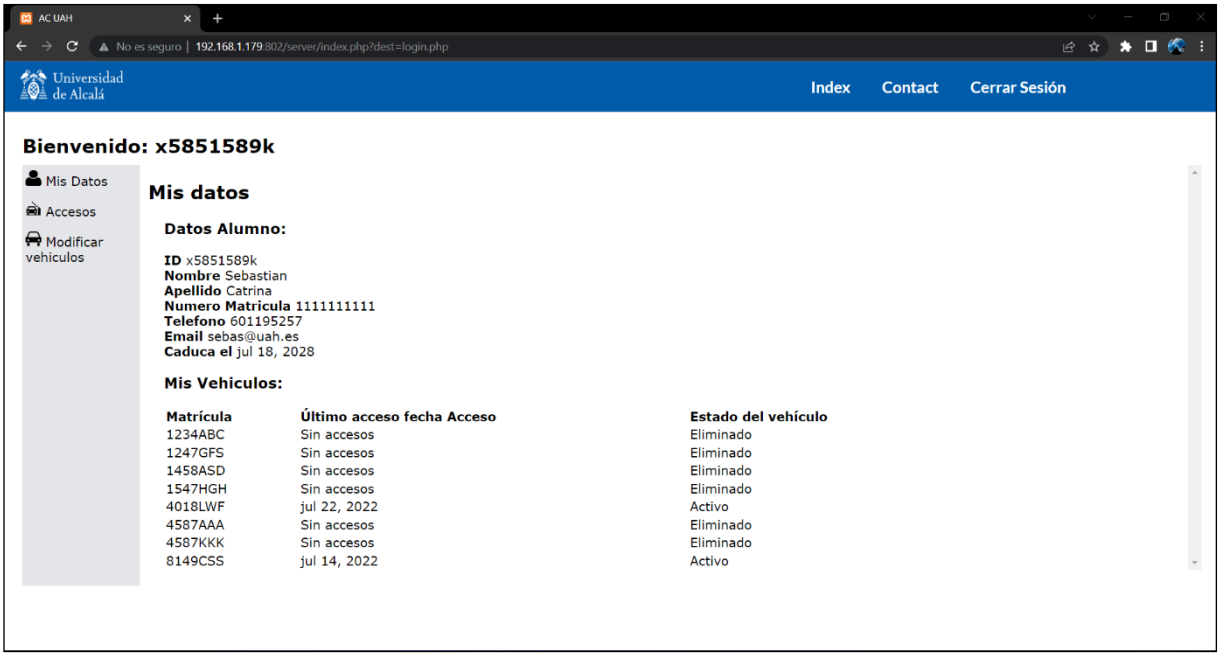

Figura 69: Página con mis datos

Vista cliente

Si se selecciona la opción de “Mis Datos”, se mostrarán los datos personales y una lista con todos los vehículos que están registrados con el DNI del usuario, la fecha del último acceso y el estado actual del vehículo, para saber si está activo o no.

Funciones

Esta vista muestra una serie de labels con los datos del alumno y una tabla con los vehículos del usuario junto con su ultimo acceso.

Para lograr esto se hace uso de la función obtenerDatosAlumno(), que llamará al Servlet GetDatosAlumno() para cargar los datos de este y la función cargarMisVehiculos (), que llamará la tabla con los vehículos del alumno cargado, haciendo uso del Servlet GetMyVehicles.

Esta pantalla muestra las opciones posibles para poder añadir y/o eliminar vehículos (Figura 70).

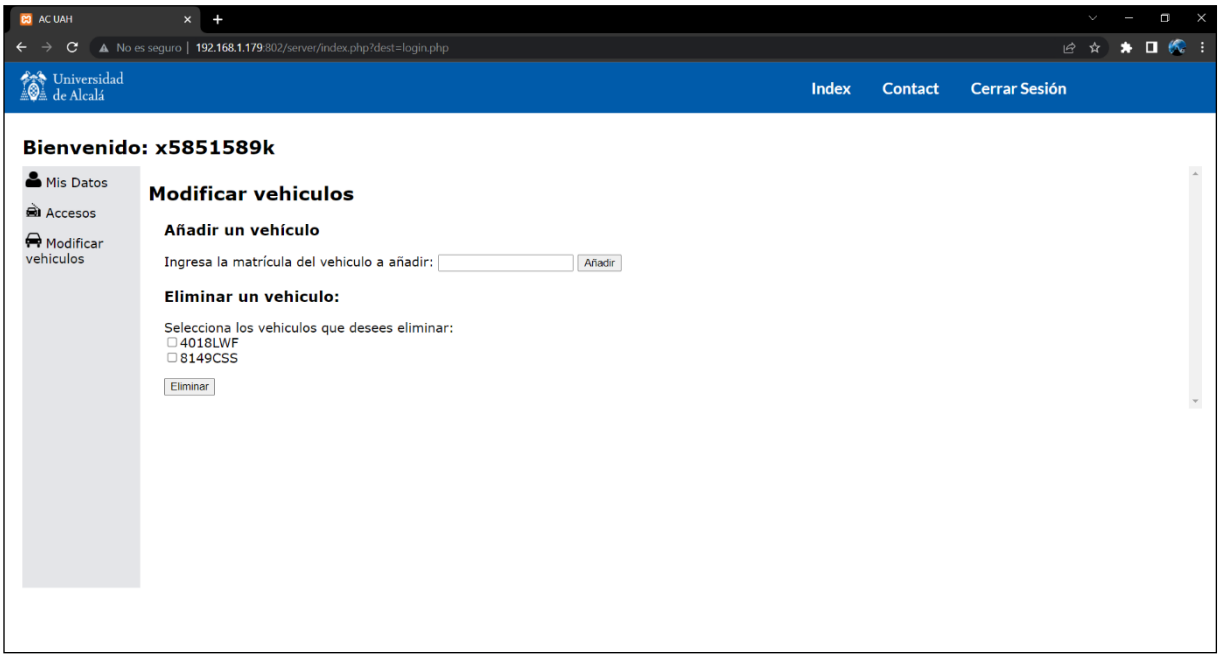

Figura 70: Página modificar vehículos

Vista cliente

La última opción permite modificar los vehículos que tiene asignados un alumno, pudiendo añadir nuevos vehículos o eliminando uno de los vehículos activos que estén registrados a su nombre.

Funciones

Esta vista tendrá dos partes, una para añadir vehículos y otra para eliminarlos. La primera tiene una entrada de texto y un botón, que, al pulsarlo, llamará a la función anadirVehiculo(). Esta función comienza comprobando que el patrón se corresponda con una posible matricula, si es así se llama al Servlet AddNewCar, que se encargará de añadir el vehículo nuevo a la base y vincularlo con el usuario,

La segunda parte mostrará una lista de vehículos, y un botón que realizará la función de desactivar todos los vehículos que se hayan seleccionado. Para eliminar el vehículo se hace uso de la form action

<form action="removeSelectedVehicles.php" method="get">, esta página lo que hará será establecer el valor activo de los vehículos seleccionados a 0.

En esta página existirán dos flujos posibles a seguir, uno para añadir vehículos (Figura 71) y otro para eliminarlos (Figura 72)

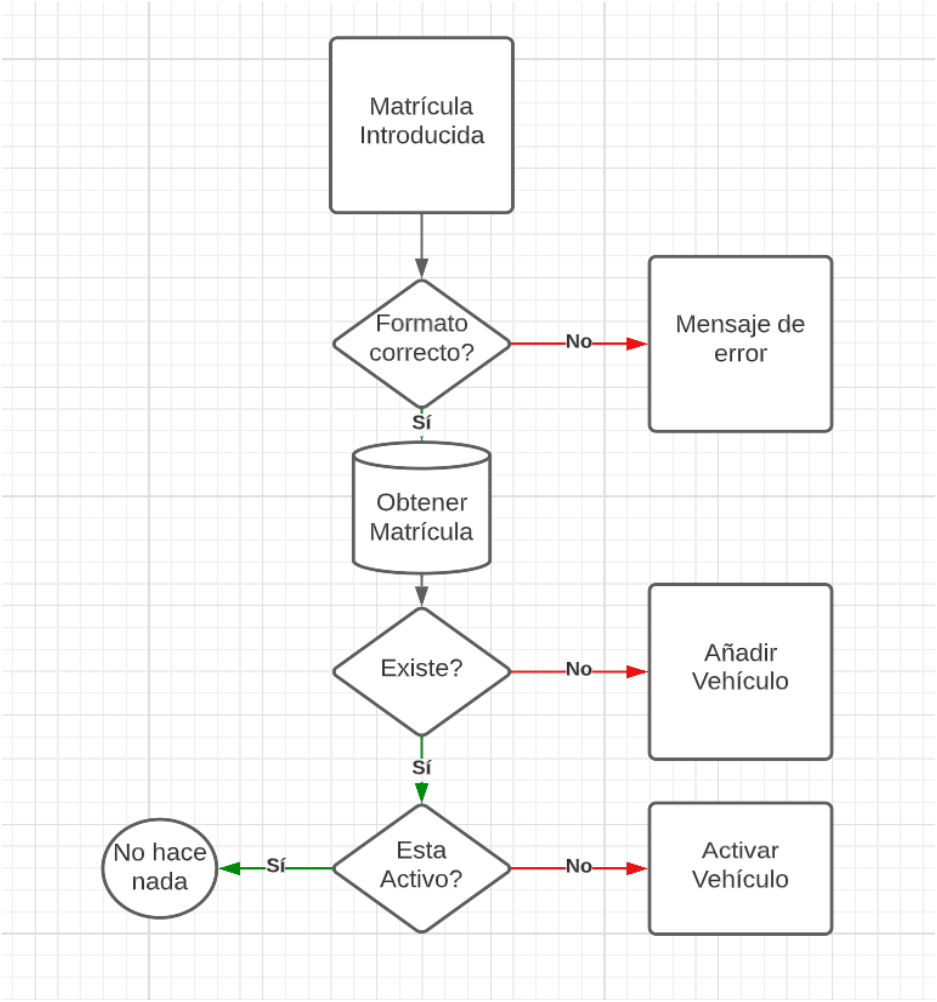

Figura 71: Añadir Vehículo

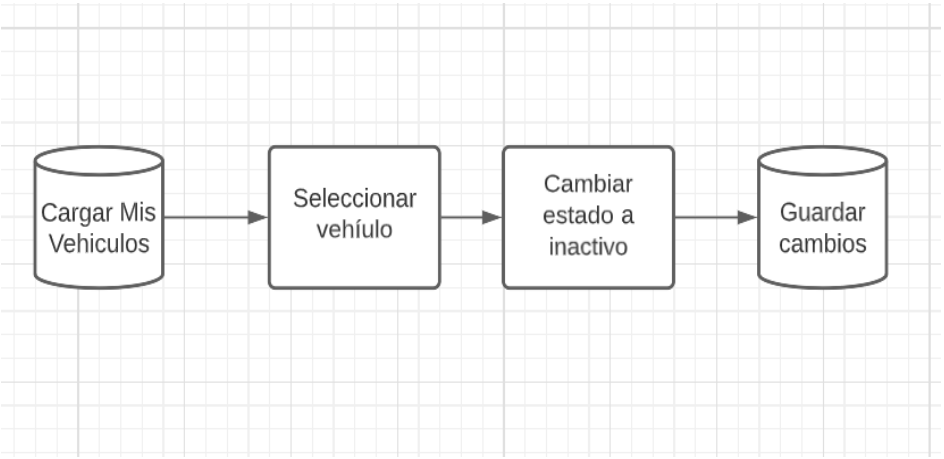

Figura 72: Eliminar vehículo

3.3 Bloque 3: Dispositivos móviles (aplicación)

Para la aplicación de móvil se ha decidido hacer uso de Android studio debido a que es un entorno que ya se había usado previamente y su funcionamiento es conocido. También tiene las ventajas de que permite usar el depurador y ver la consola del dispositivo mientras ejecutamos la aplicación. Por otro lado, permite la instalación de máquinas virtuales para probar la instalación en varios dispositivos.

Android Studio es un IDE que sirve para el desarrollo de aplicaciones nativas para Android.

Posee un potente editor de código con texto predictivo que facilita el desarrollo de código, así como un flexible sistema de compilación. También se incluyen plantillas de código que sirven para hacer uso de estructuras existentes, por ejemplo, la barra de navegación lateral que se emplea en este proyecto.

También cuenta con un emulador de Android, el cual actúa como un dispositivo completo en caso de que no se disponga de un dispositivo Android que cuente con la versión en la que se desean hacer las pruebas, aunque también permite cargar la aplicación en un teléfono real conectado al ordenador mediante un cable USB. A esto se le conoce como ADB o Android Debug Drive.

Actualmente soporta tres lenguajes de programación, Java, C y Kotlin.

3.3.1 Clases importantes

Volley

Como se verá al final de la función buscarAccesos, las peticiones a la base de datos se procesan haciendo uso de Volley.

Volley es una librería que se emplea para gestionar las peticiones externas de forma óptima.

Este componente se encarga de manejar las solicitudes a las páginas web en hilos separados, controlados directamente por la clase Volley.

En este proyecto se empleará para solicitar datos en formato JSON a unas aplicaciones PHP (Figura 73), para poder usarlo hay que crear una RequestQueue que se encargará de guardar las peticiones, una vez creado se aplica la cola al contexto actual y se añade la petición JSON que se ha creado, para que sea procesada.

```
requestQueue= Volley.newRequestQueue( context: DataActivity.this);
requestQueue.add(jsonArrayRequest);
```

Figura 73: Añadir peticiones a la cola

SharedPreferences

SharedPreferences se emplea para guardar pequeñas cantidades de datos en la memoria de la aplicación, evitando así hacer uso de otras bases de datos para poder guardar y recuperar datos organizados como clave – valor.

Estas se usan para guardar archivos de una forma permanente ya que no se pierden al cerrar la aplicación o al actualizar esta. Son una alternativa al uso de bases de datos siempre que se guardan pocos datos tales como algún ajuste o las credenciales del usuario.

En nuestro proyecto se usarán para guardar los datos de inicio de sesión del usuario junto con sus datos personales una vez iniciada la sesión, ya que así no se tendrá que volver a consultar la base de datos cada vez que acceda.

JSON Request

Se puede ver el funcionamiento de los JsonRquest a través del siguiente ejemplo (Figura 74):

```
private void buscarAccesos(String url){
    //Creamos la consulta
    JsonArrayRequest jsonArrayRequest=new JsonArrayRequest(url, new Response.Listener<JSONArray>() {
        @Override
        //Si se obtiene la respuesta, llenamos la tabla con las matrículas
        public void onResponse(JSONArray response) {

            JSONObject jsonObject = null;
            for (int i = 0; i < response.length(); i++) {
                try {
                    jsonObject = response.getJSONObject(i);
                    LayoutInflater inflater = getLayoutInflater();
                    View registro = inflater.inflate(R.layout.table_row_vehiculos,  root: null,  attachToRoot: false);
                    TextView colVehiculo=registro.findViewById(R.id.vehiculo);
                    colVehiculo.setText(jsonObject.getString( name: "matricula"));

                    tbAccesos.addView(registro);
                } catch (JSONException e) {
                    System.out.println(e.toString());
                }
            }
        }
    }, error -> System.out.println("b")
    );
    try{

        requestQueue= Volley.newRequestQueue(getContext().getApplicationContext());
```

Figura 74: Json Request

Se observa que se comienza creando el JsonArrayRequest, a la que se le pasará como parámetro la url de la página PHP de donde se realizará la consulta, dentro de esta se tiene que establecer la acción a realizar en caso de que se obtenga respuesta, esta se guarda en un JSONObject y se van obteniendo los

elementos de este para poder trabajar con ellos. El JSONRequest por sí solo no sirve de nada si no lo añadimos a una cola Volley que lo procese.

La solicitud se conecta a una url que será la que le devuelva los datos a la petición, vamos a ver un ejemplo de una de ellas para contemplar su funcionamiento (Figura 75):

GetMisVehiculos.php

```php
<?php
include "conexion.php";
$id=$_GET['id'];


$consulta = "select matricula from vehiculo where propietario='$id' and activo=1";


$resultado= $conexion -> query($consulta);

while($fila=$resultado -> fetch_array()){
    $producto[]=array_map('utf8_encode',$fila);
}
echo json_encode($producto);

$resultado -> close();
$conexion->close();

?>
```

Figura 75: Obtener vehículos PHP

Esta página comienza conectándose a la base de datos, después obtiene los datos de la url que lo ha invocado (ya que se emplea el método GET). Posteriormente se realiza la consulta y se devuelve el resultado guardado en un JSON (Figura 76).

conexion.php

```php
<?php

$hostname='localhost';
$database='controlaccesos';
$username='root';
$password='';

$conexion=new mysqli($hostname,$username,$password,$database);
if($conexion->connect_errno){
    echo "Algo ha salido mal";
}

?>
```

Figura 76: Conexión a la DB mediante PHP

Método para conectarse a una base de datos desde una página PHP, hay que establecer las credenciales de acceso previamente y después intentaremos conectarnos.

Fragment

Un Fragment representa una parte reutilizable de la IU de tu aplicación. Un fragmento define y administra su propio diseño, tiene su propio ciclo de vida y puede administrar sus propios eventos de entrada. Los fragmentos no pueden existir por sí solos, sino que deben estar alojados por una actividad u otro fragmento. Estos son especialmente útiles a la hora de hacer una aplicación modular, de forma que los elementos de estos sean independientes.

La aplicación estará formada por dos actividades principales, una de login y otra que muestra los datos haciendo uso de tres fragments, uno para los accesos, otro para las clases y otro para los datos personales.

3.3.2 Notificaciones

Las notificaciones (Figura 77) sirven para mostrar información mientras la aplicación no está en uso. Estas son una alternativa al uso de Android Toast, que muestra eventos, pero no lo hace de forma permanente, estas en cambio se muestran primero en la parte superior de forma reducida, después ofrece la posibilidad de expandir la notificación para ver más información, se pueden añadir botones con accesos rápidos o establecer acciones que se ejecutarán al pulsar la notificación.

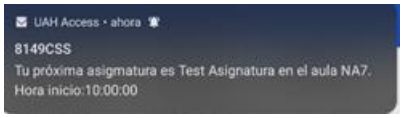

Figura 77: Ejemplo notificación

La aplicación enviara una notificación cada vez que un vehículo registrado a nuestro nombre pasa por una barrera, en caso de que sea una barrea de acceso a las instalaciones se le informara al usuario acerca de su próxima clase, en caso de que las haya. Si no hay ninguna se informará de que no tiene nada próximamente y, en caso de que sea una barrera de salida, se envía un mensaje de despedida.

En versiones superiores a Android O, no se pueden enviar notificaciones si no existe un canal creado para ello, de modo que creamos una primera función (Figura 78) que se encargará de crear este canal en caso de que la versión sea superior a esta.

```
//Creamos el canal por el cual se enviarán las notificaciones
public void createChannel(){
    //Como eso unicamente es necesario si la version de android es superior a Oreo, ponemos una
    //condición para que esto se salte en caso de que esta sea inferior
    if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O){
        NotificationChannel channel = new NotificationChannel( id: "My notification", name: "My notification", NotificationManager.IMPORTANCE_HIGH);
        NotificationManager manager = getSystemService(NotificationManager.class);
        manager.createNotificationChannel(channel);
    }
}
```

Figura 78: Crear canal de notificaciones

Una vez creado el canal de las notificaciones, que persistirá mientras la aplicación este en uso, creamos la función que se encargará de enviar la notificación (Figura 79):

```
public void sendNotification(String head, String msg){
    //Creamos el intent al que llamaremos para que se abra la aplicación al hacer click
    Intent intent = new Intent( packageContext: this, DataActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
    PendingIntent pendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, intent,  flags: 0);

    //Se crea el constructor para la notificación
    NotificationCompat.Builder builder =new NotificationCompat.Builder( context: DataActivity.this, channelId: "My notification");
    //Establecemos el título
    builder.setContentTitle(head);
    //Establecemos e cuerpo del mensaje
    builder.setContentText(msg);
    //Ponemos un icono para la notificación
    builder.setSmallIcon(R.drawable.icon_email);
    //Permitimos que la notificación se quite sola trás un tiempo
    builder.setAutoCancel(true);
    //Establecemos la prioridad de la notificación
    builder.setPriority(NotificationCompat.PRIORITY_HIGH);
    //Permitimos que el contenido de la notificación se muestre en la pantalla de bloqueo
    builder.setVisibility(NotificationCompat.VISIBILITY_PUBLIC);
    //Esta parte esta porque el texto de la notificación se recorda para caber en esta, estableciendo esto
    //se permite expandirla para poder mostrarla completa
    builder.setStyle(new NotificationCompat.BigTextStyle()
            .bigText(msg));
    //Establecemos la accion que se ocurirá al hacer click en la notificación, este intent
    //esta configurado para abrir la aplicación
    builder.setContentIntent(pendingIntent);
    //Creamos el manager para la notificación
    NotificationManagerCompat managerCompat = NotificationManagerCompat.from(DataActivity.this);
    //Enviamos la notificación
    managerCompat.notify( id: 1,builder.build());
}
}
```

Figura 79: Enviar notificación

La función encarga de enviar la notificación comienza creando un Intent. Un Intent es un objeto de mensajería que se puede usar para solicitar una acción de otro componente de una aplicación, pero para poder ejecutar el Intent desde fuera de la aplicación tendremos que configurar un PendingIntent, que se usa para darle a una actividad externa, siendo esta la notificación, los permisos para ejecutar el Intent.

Para configurar la notificación, primero hay que crear el builder de esta, en el que se configurarán todos los parámetros de la notificación:

Title: Texto que se mostrará en la notificación, este será el tópico del mensaje MQTT, pero se puede pasar cualquier valor como parámetro.

Text: Texto que formará el cuerpo de la notificación, si este es demasiado largo se recortará para que, entre mejor en el banner, aquí se verá reflejado el mensaje que se ha publicado mediante MQTT ya que en este proyecto este informa directamente de la próxima clase que tiene el usuario con la matricula que pasó.

SetSmallIcon: Icono que mostrará la notificación.

SetAutoCancel: Hace que la notificación se elimine directamente cuando se pulsa sobre ella, ya que hacer esto abrirá la aplicación.

SetPriority: Establece la prioridad de la notificación, este método ya no sirve en versiones más recientes, pero como tuvimos que usar una antigua por la compatibilidad del MQTT usaremos esta. Si se trabaja en una api superior a 26 habrá que usar setImportance().

SetVisibility: Establece la privacidad de la notificación, en nuestro caso hemos decidido que sea publica para que se puedan ver desde la pantalla de bloqueo.

SetStyle: Dentro de esta se establece el estilo de la notificación expandida, para que esta pueda mostrar el texto sin recortar.

SetContentIntent: aquí se introducirá el pending intent para que se pueda abrir la aplicación desde la notificación.

Una vez ajustado todo se construye la notificación usando build y la se envía haciendo uso de notify.

3.3.3 Consultas

Para realizar consultas en la base de datos se ha decidido hacer uso de páginas PHP que devuelven archivos JSON, para esto crearemos el JsonArrayRequest. El problema es que no se puede ejecutar directamente, por eso se hará uso de la función Volley.

Estas consultas se emplearán para cargar el historial de accesos, cargar las clases que el usuario tendrá en el día actual y todas las clases a las que esta apuntado, otra que carga los datos del usuario y otra que carga sus vehículos para suscribirse a ellos (Figura 80).

conexion.php

```php
<?php

$hostname='localhost';
$database='controlaccesos';
$username='root';
$password='';

$conexion=new mysqli($hostname,$username,$password,$database);
if($conexion->connect_errno){
    echo "Algo ha salido mal";
}

?>
```

Figura 80: Conexión con la Base

Esta página será la encargada de conectarse con la base de datos, una vez conectado ya se pueden realizar las consultas.

Vamos a ver cómo es la página PHP encargada de proporcionar el JSON (Figura 81):

Figura 81: Página PHP que devuelve un JSON

En esta página se comienza obteniendo los parámetros del enlace y prepara el select que se hará en la base de datos.

Después se ejecuta la consulta en la base a la que se habrá conectado previamente haciendo uso de "conexión.php", posteriormente habrá que convertir la respuesta en JSON y devolverla al programa.

3.3.4 MQTT

Para utilizar el servicio MQTT y poder conectarse a Mosquitto se hará uso de la biblioteca **Eclipse Paho Android.**

Paho Android Service es una biblioteca MQTT escrita en Java que se usa para desarrollar aplicaciones en Android.

El proyecto Paho se creó para proporcionar implementaciones confiables de código abierto de protocolos de mensajería abiertos y estándar destinados a aplicaciones nuevas, existentes y emergentes para Machine-to-Machine (M2M) e Internet de las cosas (IoT).

Para poder usar esta librería debemos instalarlo añadiendo las dependencias y los repositorios dentro del Gradle. Una vez instalado hay que añadir el servicio al AndroidManifest.xml y garantizar una serie de permisos que permitirán recibir las notificaciones.

Una vez hecho esto habrá que dirigirse a la clase en la que se quiera hacer uso de este servicio, en esta se creó el cliente MQTT dentro de la función init() (Figura 82) haciendo uso del siguiente fragmento de código:

```
private void init(){
    //Creamos el cliente y nos conectamos
    String clientId = MqttClient.generateClientId();
     client =
            new MqttAndroidClient(this.getApplicationContext(), serverUri,
                    clientId);
    try {
        IMqttToken token = client.connect();
        token.setActionCallback(new IMqttActionListener() {
            @Override
            //Si nos podemos conectar, buscamos los vehiculos del usuario y nos suscribimos
            public void onSuccess(IMqttToken asyncActionToken) {
                // Nos hemos conectado
                Log.d(TAG,  msg: "Conectado al servidor MQTT");
                buscarAccesos( url: "http://"+data.ip+"/Server/db/GetMisVehiculos.php?id="+ID);
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Log.d(TAG,  msg: "No se ha podido conectar al servidor MQTT");
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Figura 82: Iniciar conexión MQTT Android

Esta función llama, en caso de que se haya logrado establecer la conexión con el servidor, a una nueva que buscará las matrículas de los vehículos pertenecientes a los usuarios y se suscribirá a estas.

Se puede observar que esta función crea un JSONRequest para obtener los datos de los vehículos y se suscribe a todos los valores recibidos (Figura 83):

```
private void buscarAccesos(String url){
    //Se solicitan los datos
    JsonArrayRequest jsonArrayRequest=new JsonArrayRequest(url, new Response.Listener<JSONArray>() {
        @Override
        //Si se recibe respuesta
        public void onResponse(JSONArray response) {
            JSONObject jsonObject = null;
            //Se itera por las matrñículas recibidas
            for (int i = 0; i < response.length(); i++) {
                try {
                    //Nos suscribimos a las matriculas que se detecten
                    jsonObject = response.getJSONObject(i);
                    System.out.println(jsonObject.getString( name: "matricula"));
                    //Nos suscribimos tanto a las matriculas en mayuscula como en minúscula
                    sub(jsonObject.getString( name: "matricula").toLowerCase());
                    sub(jsonObject.getString( name: "matricula").toUpperCase());

                } catch (JSONException e) {
                    System.out.println(e.toString());
                }
            }
        }
    }, error -> System.out.println("b")
    );

    requestQueue= Volley.newRequestQueue( context: DataActivity.this);
    requestQueue.add(jsonArrayRequest);
}
```

Figura 83: Función buscarAccesos()

Con las matrículas obtenidas, se suscribirá a todas estas para poder enviar las notificaciones cuando algún vehículo llegue a la barrera, para esto se hace uso de la función sub() (Figura 84):

```
private void sub(String topic){
    //Establecemos la qos a 0 ya que no nos interesa recibir las notificaciones después de
    //haber accedido, si no se ha podido entregar en el momento en el que se ha enviado deja
    //de tener importancia.
    int qos = 0;
    try {
        //Nos suscribimos
        client.subscribe(topic, qos);
        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {

            }
            //Accion que se realiza al recibir un mensaje
            @Override
            public void messageArrived(String topic, MqttMessage message) throws Exception {
                //Enviamos la notificación
                 sendNotification(topic,message.toString());

                 Log.d(topic,message.toString());
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {

            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Figura 84: Función sub()

Esta función se suscribe al tópico que le pasamos como parámetro y modifica la acción que se realiza en caso de que se reciba un acceso para que se envié la notificación haciendo uso de la función sendNotification().

3.3.5 Diseño

Al iniciar la aplicación siempre se mostrará una pantalla de carga durante medio segundo aproximadamente. Esta sirve para que tenga tiempo de comprobar si el usuario ha iniciado sesión. En caso de que no lo haya hecho se mostrará esta pantalla en la que se le solicita los datos de acceso (Figura 85) al usuario. Se ha decidido no incluir la opción de registrarse desde dispositivos móviles ya que la finalidad de esta aplicación es únicamente la consulta de datos.

Ambas entradas tienen verificaciones para comprobar que se hayan introducido todos los datos y muestra una notificación Toast en caso de que falte algo. La comprobación realizada para ver que los datos se hayan introducido correctamente y se inicie sesión tiene el mismo funcionamiento que en la versión Web.

Si todo es correcto se muestra un mensaje de bienvenida y se muestra la lista de accesos recientes.

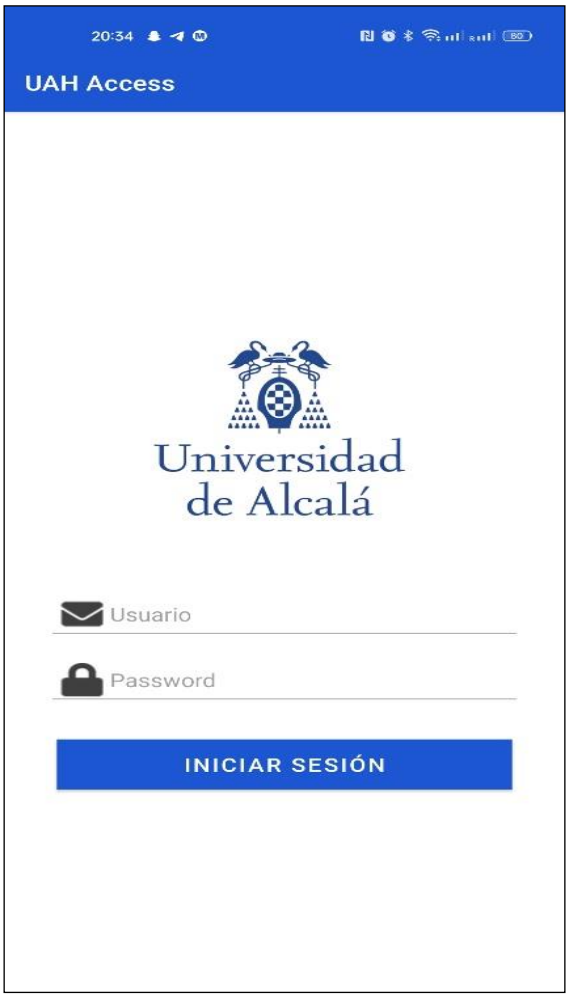

Figura 85: Pantalla de Login Android

Dentro de la aplicación se dispondrá de una barra lateral (Figura 86) que permitirá cambiar entre los diferentes fragmentos disponibles. En la parte superior se muestra el nombre y los apellidos del usuario junto con su correo. Se puede ver que las opciones que hay actualmente son consultar el historial de accesos, las clases y los datos del alumno, pero se pueden añadir más fragmentos fácilmente en caso de que se quieran tratar más datos o añadir nuevas funciones.

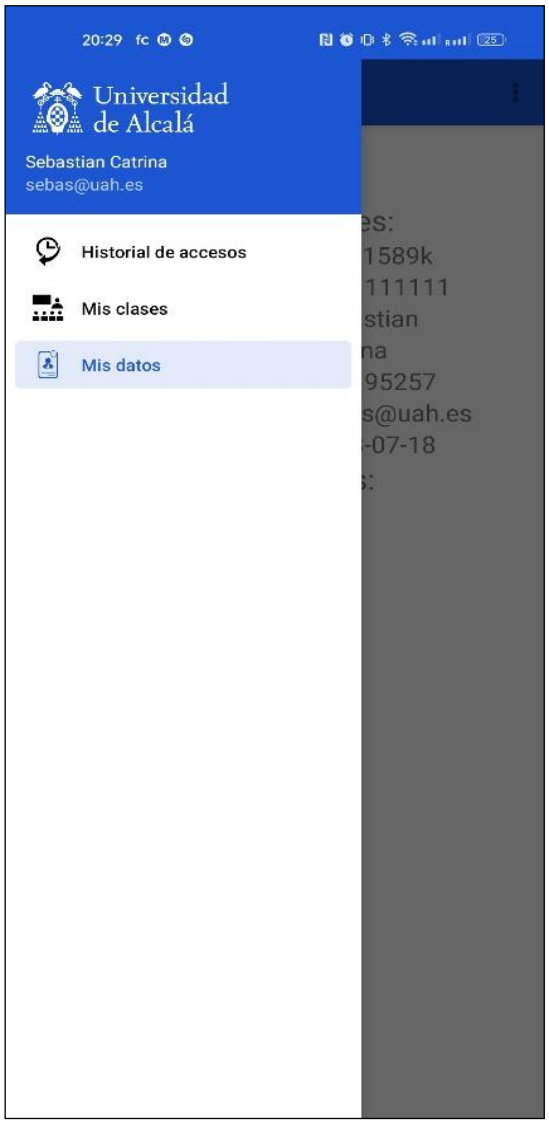

Figura 86: Barra navegación lateral Android

Esta pantalla muestra el historial de accesos de los vehículos del usuario (Figura 87), se ve que la tabla es muy similar a la que está presente en la página web, para esta se ha creado una tabla y un elemento de la clase table row. Este elemento se irá repitiendo y añadiendo a la tabla mientras se van obtenido diferentes filas de la consulta a la base de datos.

Todos estos elementos se van añadiendo a un ScrollView para que se pueda deslizar a través de los accesos sin que se muevan los títulos de la tabla.

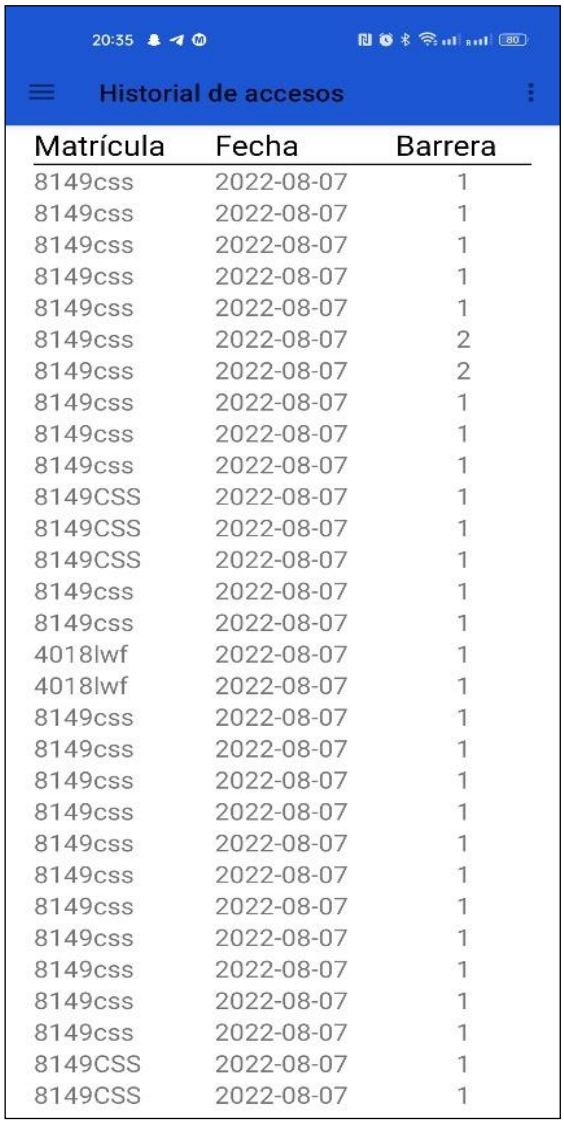

Figura 87: Historial de accesos Android

En este fragmento se muestran las asignaturas que tendrá el usuario ese día y todas en la que está matriculado (Figura 88). Para esto se conecta a la base de datos y se consultarán las clases en las que está matriculado cada alumno junto con el horario de esta. Para mostrarlo igual que en el apartado anterior se irán creando las columnas una a una y se añadirán dentro del ScrollView.

La primera tabla muestra una lista con las asignaturas que se tienen a lo largo de ese día y que aún no han terminado.

La segunda selecciona todas las asignaturas en las que se ha matriculado el alumno.

El día de la semana se devuelve en formato numérico, así que se hace uso de un switch para convertirlo a letras y que resulte más fácil de leer.

Figura 88: Mis Clases Android

En este fragmento se mostrarán los datos personales del alumno (Figura 89), para eso se han cargado los datos personales que se han guardado dentro de las SharedPreferences cuando se inició sesión. Después se realizará un JSON Request para obtener los vehículos del alumno.

Figura 89: Datos personales Android

4. Conclusión

Si nos centramos en las conclusiones con respecto al proyecto, se han logrado crear e implementar todos los elementos inicialmente pensados, habiendo creado un sistema independiente para el reconocimiento de matrículas que notifica los accesos a cualquier programa, un servidor compatible con cualquier base de datos sin necesidad de realizar grandes modificaciones y una aplicación Android capaz de recibir datos y enviar notificaciones en tiempo real.

En cuanto a las conclusiones técnicas, se ha logrado una calidad de obtención aceptable, haciendo uso de algoritmos optimizados para detectar imágenes en condiciones de luz uniforma, dando problemas en caso de que haya sombras presentes al no poder detectar correctamente el contorno de la imagen. Por otra parte, el servidor ha respondido con bastante rapidez a las pruebas realizadas, sin ofrecer ninguna clase de problemas o tiempos de carga. En cambio, la aplicación de Android sí que se observa algo más lenta, tardando unos segundos en cargar datos, aunque esto puede deberse al bajo rendimiento de los dispositivos empleados o a la velocidad de conexión.

Gracias a este trabajo se ha aprendido el funcionamiento de la visión artificial o, al menos, una ligera idea de este, ya que por lo que se ha observado, este es un área muy extensa y muy complejo, que posee una enorme cantidad de algoritmos y posibilidades para realizar cada tarea. También se ha mejorado el conocimiento acerca del funcionamiento de bases de datos y la integración de estas con distintas aplicaciones y dispositivos. Por último, se ha mejorado la capacidad de creación de páginas web y aplicaciones móviles interconectadas entre ellas.

5. Bibliografía

Aquí se muestra una lista con todos los enlaces que se han consultado separados por categorías

Instalar Mosquitto

https://www.youtube.com/watch?v=cL6n3wafWEQ

Subcadenas

https://j2logo.com/python/buscar-subcadena-en-python/

https://docs.python.org/es/3/library/re.html#functions

https://www.delftstack.com/es/howto/python/extract-substring-from-a-string-in-python/

https://www.journaldev.com/23763/python-remove-spaces-from-string

https://pynative.com/python-regex-capturing-groups/#:~:text=A%20group%20is%20a%20part,'%2C%20and%20't'.

Today date

https://stackoverflow.com/questions/5046771/how-to-get-todays-date#:~:text=Code%20To%20Get%20Today's%20date%20in%20any%20specific%20Format&text=getTime()%3B%20String%20todaysdate%20%3D%20dateFormat,Today's%20date%20%3A%20%22%20%2B%20todaysdate)%3B

Password

https://developer.mozilla.org/es/docs/Web/HTML/Element/input/password

HTML

https://www.w3schools.com/tags/tag_center.asp

https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_center_css

https://elblogdeliher.com/como-mostrar-los-acentos-y-las-n-en-html/

https://www.w3schools.com/css/css_font_size.asp

https://www.w3schools.com/tags/tag_font.asp

https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_font_color_css

Login

https://es.stackoverflow.com/questions/78414/como-mostrar-los-datos-de-una-base-de-datos-en-una-tabla-en-html-y-php

https://www.youtube.com/watch?v=mZG7VvHkOjI

https://vaidrollteam.blogspot.com/2021/03/login-basico-con-php-html-xampp-mysql.html

https://vaidrollteam.blogspot.com/2021/03/crear-login-basico-con-sesiones-en-php.html

Sesiones

https://diego.com.es/sesiones-en-php

Consola PHP

https://www.delftstack.com/es/howto/php/print-to-console-php/

PHP en Tomcat
https://www.quora.com/Can-Tomcat-run-PHP

https://www.youtube.com/watch?v=WQ3epZ4t3Zs

ACCES CONTROL ALLOW ORIGIN
https://medium.com/@dtkatz/3-ways-to-fix-the-cors-error-and-how-access-control-allow-origin-works-d97d55946d9

Cambiar un label desde JavaScript
https://stackoverflow.com/questions/4488714/change-label-text-using-javascript

https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Traversing_an_HTML_table_with_JavaScript_and_DOM_Interfaces

Forms html
https://www.htmlquick.com/es/tutorials/forms/2.html

Strings match JavaScript
https://www.w3schools.com/jsref/jsref_match.asp

https://es.stackoverflow.com/questions/48115/como-poner-una-matricula-en-java-segura

Leer respuestas base de datos
https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=613:ejemplo-consulta-php-mysql-select-bucle-while-mysqlifetcharray-recorrer-resultados-query-cu00842b&catid=70&Itemid=193

Inputs html
https://www.htmlquick.com/es/reference/tags/input-checkbox.html

https://www.delftstack.com/es/howto/javascript/get-html-form-value/#:~:text=En%20JavaScript%2C%20podemos%20usar%20el,datos%20sin%20procesar%20del%20usuario.

Cerrar sesión
https://www.web-dev-qa-db-esp.com/es/php/php-destruccion-de-sesion-en-el-boton-cerrar-sesion/942658182/#:~:text=El%20proceso%20es%3A%20%2D%20Haga%20clic,al%20%C3%ADndice%20de%20nivel%20superior.

Arduino
https://www.luisllamas.es/como-emular-un-teclado-o-un-raton-con-arduino/

https://naylampmechatronics.com/blog/10_tutorial-de-arduino-y-sensor-ultrasonico-hc-sr04.html

Android studio Notifications
https://www.youtube.com/watch?v=B5dgmvbrHgs

https://openwebinars.net/blog/como-hacer-notificaciones-push-en-android-facil/

https://www.youtube.com/watch?v=4BuRMScaaI4&ab_channel=EasyTuto

https://www.youtube.com/watch?v=B5dgmvbrHgs

https://developer.android.com/training/notify-user/build-notification?hl=es-419 - java

https://www.tutorialspoint.com/android/android_notifications.htm

https://developer.android.com/reference/android/app/Notification.Builder

Android Studio MQTT

https://zoransasko.medium.com/sending-and-receiving-mqtt-messages-in-android-and-ios-applications-45e85d23473f

https://solace.com/blog/event-driven-kotlin-android-app-mqtt-solace-pubsub/

https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service/

https://www.youtube.com/watch?v=0LRqw-wGQds&ab_channel=occ

https://www.youtube.com/watch?v=6A8rtMNTNqc&list=PL7tE2OhgQCeJyM4dHKJjdKgpvzxmijt2C&index=6

https://www.youtube.com/watch?v=34Pt85l9uW0&list=PL7tE2OhgQCeJyM4dHKJjdKgpvzxmijt2C&index=3&t=22s

https://www.youtube.com/watch?v=0LRqw-wGQds&ab_channel=occ

https://github.com/zsasko/mqtt-android-sample

https://www.youtube.com/watch?v=BBLqa2Wh6nQ&ab_channel=YourCyberMentor

https://github.community/t/mqtt-problem-android-app/12688/2

https://github.com/eclipse/paho.mqtt.android/issues/321

https://github.com/eclipse/paho.mqtt.android/issues/420

Cambiar vista Android

https://www.tutorialesprogramacionya.com/kotlinparaandroidya/detalleconcepto.php?punto=11&codigo=59&inicio=0

Error MQTT

https://github.com/eclipse/paho.mqtt.android/issues/321

https://github.community/t/mqtt-problem-android-app/12688/2

Kotlin MYSQL

https://www.youtube.com/watch?v=I33dlFY3qWU&ab_channel=YouQuince

https://www.youtube.com/watch?v=yh4URGTGmBQ

HTTP JSON Request en Android

https://www.android--code.com/2020/10/android-kotlin-volley-jsonobjectrequest.html

https://stackoverflow.com/questions/64817901/http-json-request-in-android-r-and-kotlin

https://getridbug.com/android/android-kotlin-volley-how-to-send-json-data/

Kotlin web app

https://www.journaldev.com/17957/kotlin-web-application-tutorial

MYSQL Android studio
https://www.youtube.com/watch?v=34Pt85l9uW0&list=PL7tE2OhgQCeJyM4dHKJjdKgpvzxmijt2C&index=3&t=22s

https://www.youtube.com/watch?v=34Pt85l9uW0&list=PL7tE2OhgQCeJyM4dHKJjdKgpvzxmijt2C&index=3&t=229s

https://www.youtube.com/watch?v=34Pt85l9uW0&ab_channel=DEVELOPERU

https://developpaper.com/android-studio-servlet-mysql-realizes-login-and-registration/

Llenar tabla desde Android con MySQL
https://www.youtube.com/watch?v=xLuKZIpQYwI&ab_channel=ProgramadorNovato

https://github.com/programadornovato/AndroidMysql/commit/3736cd8b97e54808538a62e5e0f8e13a36e24c48#diff-4536e1f2f94eb1c79a477da44c62fa563a2ceee1cf27b4449a81277f9be3800a

Menús Android
https://developer.android.com/guide/topics/ui/menus

Layout inflater para tablas
https://stackoverflow.com/questions/3477422/what-does-layoutinflater-in-android-do

Scrollable table layout
https://stackoverflow.com/questions/6513718/how-to-make-a-scrollable-tablelayout

Volley
https://stackoverflow.com/questions/28172496/android-volley-how-to-isolate-requests-in-another-class

Scroll layout
https://stackoverflow.com/questions/6513718/how-to-make-a-scrollable-tablelayout

Shared preferences
https://stackoverflow.com/questions/11741270/android-sharedpreferences-in-fragment

Obtener texto de imágenes
https://omes-va.com/reconocimiento-de-matriculas-vehiculares-opencv-pytesseract-ocr-python/

https://www.marindelafuente.com.ar/reconocimiento-de-matriculas-con-raspberry-pi-y-opencv/

https://github.com/Maleehak/Car-number-plate-recognition-using-OpenCV

https://www.geeksforgeeks.org/license-plate-recognition-with-opencv-and-tesseract-ocr/

https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/

https://www.section.io/engineering-education/license-plate-detection-and-recognition-using-opencv-and-pytesseract/

https://nanonets.com/blog/ocr-with-tesseract/

https://ec2-cli.com/es/ocr-con-python-opencv-y-pytesseract/

https://www.delftstack.com/es/howto/python/opencv-invert-image/

https://stackoverflow.com/questions/44619077/pytesseract-ocr-multiple-config-options

https://muthu.co/all-tesseract-ocr-options/

https://parzibyte.me/blog/2019/05/11/instalar-tesseract-ocr-windows-10/ -Descargar_Tesseract_OCR

https://github.com/UB-Mannheim/tesseract/wiki

https://circuitdigest.com/microcontroller-projects/license-plate-recognition-using-raspberry-pi-and-opencv

https://learnopencv.com/automatic-license-plate-recognition-using-deep-learning/

Configurar pytesseract ocr
https://stackoverflow.com/questions/44619077/pytesseract-ocr-multiple-config-options

Pattern detection en pytesseract ocr
https://stackoverflow.com/questions/31874393/tesseract-ocr-force-pattern

https://manpages.ubuntu.com/manpages/bionic/man1/tesseract.1.html

https://stackoverflow.com/questions/17209919/tesseract-user-patterns

https://tesseract-ocr.github.io/tessdoc/APIExample-user_patterns.html

HARR cascade
https://medium.com/analytics-vidhya/license-plate-detection-from-car-images-by-using-harr-cascade-and-open-cv-255f296557cc

https://www.geeksforgeeks.org/erosion-dilation-images-using-opencv-python/

Binarizar imagen
https://programmerclick.com/article/3598728662/

Filtrado de imágenes
https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed

Encontrar contornos
https://stackoverflow.com/questions/25733694/process-image-to-find-external-contour

Aproximar a polígono
https://www.programcreek.com/python/example/89328/cv2.approxPolyDP

https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#ga0012a5fdaea70b8a9970165d98722b4c

Homografía

https://www.pythonpool.com/cv2-findhomography/

https://learnopencv.com/homography-examples-using-opencv-python-c/

https://docs.opencv.org/3.4/d7/dff/tutorial_feature_homography.html

https://www.geeksforgeeks.org/python-opencv-object-tracking-using-homography/

https://www.educba.com/opencv-findhomography/

https://www.youtube.com/watch?v=5LGkM3BtcCI

https://docs.opencv.org/4.x/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87

Obtener tamaño imagen

https://www.tutorialkart.com/opencv/python/opencv-python-get-image-size/

Encontrar media para el centro de masas

https://geekflare.com/es/python-find-mean-median-and-mode/

https://www.delftstack.com/es/api/numpy/python-numpy-mean/

Otros TFG

https://zaguan.unizar.es/record/60549/files/TAZ-TFG-2017-140.pdf

https://oa.upm.es/51869/1/TFG_JORGE_NAVACERRADA.pdf

https://oa.upm.es/53515/1/TFG_ALVARO_AZOFRA_DE_LAS_HERAS.pdf

Diseño diagramas

https://lucid.app/

6. Anexos

6.1 Manual de usuario

Ya que este sistema está formado de múltiples componentes, habrá que instalar cada uno por separado para que pueda funcionar. Se va a indicar el orden el en que se deben instalar para evitar incompatibilidades

El primero de ellos será el servidor Mosquitto (o cualquier servicio MQTT a elegir), ya que este no depende de ningún otro, posteriormente se tendrá que instalar la base de datos, igual que el paso anterior, ya que se dispone de una copia en SQL de esta se puede hacer uso de cualquiera, lo único que se necesita es obtener el JDBC Driver de la base correspondiente para poder trabajar con ella.

Para la base de datos habrá que importar las tablas creadas y configurar los usuarios para permitir el acceso remoto.

El próximo paso sería desplegar el war en el servidor Tomcat, si se ha modificado la base de datos habrá que volver a exportar el proyecto modificando los enlaces de accesos creados en la clase datos.

Ahora que ya está todo el servidor creado se puede desplegar tanto la página web como la aplicación de Android, lo primero es desplegar la página web, para esto hay que guardar los archivos dentro de la carpeta htdocs de XAMPP y, una vez hecho esto, ya estará disponible para usar la aplicación Android, que se puede instalar directamente en los dispositivos móviles.

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR