

UNIVERSIDAD DE CÓRDOBA

Manual de Usuario

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE
RECONOCIMIENTO FACIAL PARA CONTROL DE
ACCESO

Autor: Francisco Javier Ortiz Herrerías

Tutor: Joaquín Olivares Bueno

septiembre de 2025

Índice general

1. Introducción	4
2. Instalación de herramientas necesarias	5
2.1. Instalación de Python	5
2.2. Instalación de Visual Studio Code	6
2.3. Instalación de Arduino IDE	6
2.4. Instalación de Android Studio	7
2.5. Instalación de Flutter y Dart	7
2.6. Instalación de compiladores de C++ (Visual Studio Build Tools) . . .	9
3. Instalación del sistema	10
3.1. Instalación del servidor	10
3.2. Instalación del dispositivo ESP32-CAM	12
3.3. Instalación de la aplicación móvil	15
3.4. Configuración de red y direcciones IP	17
4. Puesta en marcha	18
4.1. Encendido del ESP32-CAM	18
4.2. Inicio del servidor FastAPI	19
4.3. Conexión de la aplicación móvil	20
5. Funcionamiento del sistema	21
5.1. Interfaz web del sistema	21
5.2. Interfaz móvil	23
5.2.1. Historial de accesos	23
5.2.2. Cámara en vivo	24
5.2.3. Rostros enrolados	25
6. Resolución de problemas comunes	27
6.1. Emulador de Android “congelado” o con imagen deformada	27
6.2. Error con la cámara del ESP32-CAM	29

6.3. ESP32-CAM no conectado en Arduino IDE	29
6.4. La aplicación se queda cargando en el historial de accesos	30
6.5. Monitor Serie muestra caracteres extraños	31
6.6. Problemas durante la instalación de dependencias	32
6.7. SDK de Android incompleto en Android Studio	33
6.8. Error al ejecutar <code>flutter pub get</code> (conflicto con intl)	34
6.9. Dart SDK not configured	34
7. Preguntas frecuentes (FAQ)	36

Capítulo 1

Introducción

Este manual tiene como objetivo guiar al usuario en la instalación y el uso del sistema de control de acceso desarrollado en el Trabajo Fin de Grado.

El sistema combina un dispositivo **ESP32-CAM**, un servidor central basado en **FastAPI** y una interfaz de usuario que puede ser utilizada tanto desde una aplicación móvil en **Flutter** como desde una interfaz web accesible mediante navegador.

La instalación y el manejo del dispositivo y de la aplicación se realizan mediante interfaces gráficas sencillas, mientras que la puesta en marcha del servidor requiere la ejecución de un comando desde la terminal.

Aun así, todos los procedimientos se explican de forma clara y paso a paso, de modo que cualquier persona, incluso sin experiencia técnica avanzada, pueda reproducirlos sin dificultad.

Capítulo 2

Instalación de herramientas necesarias

Antes de comenzar con la instalación del sistema, es importante disponer de varias herramientas de software que se utilizarán en distintas fases: **Visual Studio Code** para el servidor, **Arduino IDE** para el dispositivo ESP32-CAM y **Android Studio** (o, alternativamente, Visual Studio Code con extensiones de Flutter) para la aplicación móvil. Además, será necesario instalar **Flutter y Dart** y los compiladores de **C++**.

2.1. Instalación de Python

Para ejecutar el servidor FastAPI y las librerías necesarias, es imprescindible tener instalado **Python 3.9 o superior**.

La descarga oficial está disponible en:

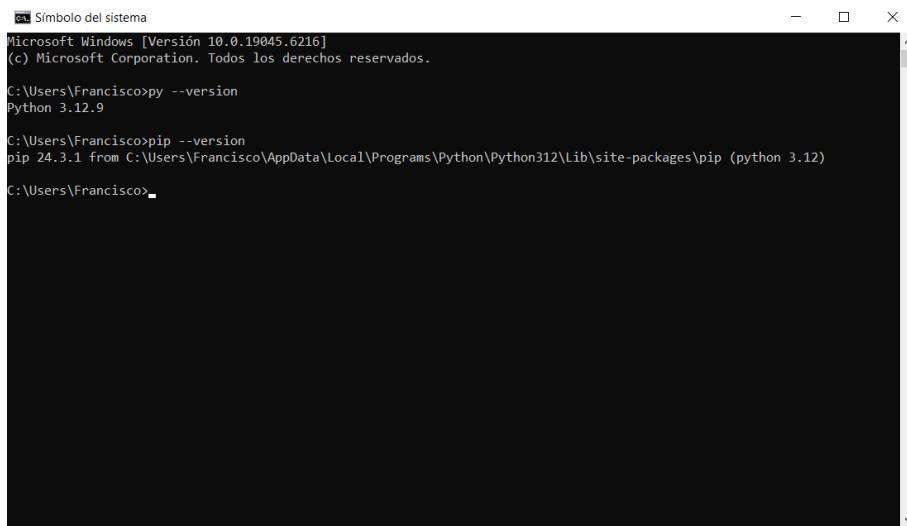
<https://www.python.org/downloads/>

Durante la instalación en Windows, se recomienda marcar la opción “**Add Python to PATH**” para evitar problemas al ejecutar los comandos desde la terminal.

Una vez finalizada la instalación, puede verificarse que Python y el gestor de paquetes pip están disponibles ejecutando en la terminal:

```
python --version  
pip --version
```

El resultado debería mostrar la versión instalada de Python y de pip, confirmando que la instalación fue correcta.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.6216]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Francisco>py --version
Python 3.12.9

C:\Users\Francisco>pip --version
pip 24.3.1 from C:\Users\Francisco\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)
C:\Users\Francisco>
```

Figura 2.1: Comprobación de la versión de Python y pip en la terminal de Windows.

2.2. Instalación de Visual Studio Code

Visual Studio Code es el entorno que se utilizará para ejecutar el servidor central. Puede descargarse desde la página oficial:

<https://code.visualstudio.com/download>

Una vez descargado, basta con:

1. Abrir el instalador.
2. Aceptar la licencia.
3. Seleccionar la carpeta de instalación.
4. Pulsar en *Siguiente* hasta finalizar.

2.3. Instalación de Arduino IDE

Arduino IDE será necesario para cargar el firmware en el ESP32-CAM. Disponible en:

<https://www.arduino.cc/en/software>

Tras la descarga, el proceso es sencillo: abrir el archivo, aceptar la licencia, pulsar en *Next* y finalizar con *Finish*.

2.4. Instalación de Android Studio

La aplicación móvil desarrollada en Flutter se compila y ejecuta desde Android Studio. Descarga oficial:

<https://developer.android.com/studio>

Durante la instalación se pedirá aceptar la licencia y seleccionar la carpeta de destino. En el primer arranque, Android Studio pedirá instalar componentes adicionales (SDK de Android y emuladores). Basta con aceptar las opciones por defecto.

2.5. Instalación de Flutter y Dart

La aplicación móvil está desarrollada con Flutter, por lo que es necesario instalar este framework y su SDK.

1. Acceder a: <https://docs.flutter.dev/get-started/install>
2. Descargar la versión correspondiente al sistema operativo.
3. Descomprimir el archivo en una carpeta sin espacios ni caracteres especiales, por ejemplo: C:\Users\TUNOMBRE\dev\flutter
4. Configurar la variable de entorno PATH para incluir la carpeta flutter\bin.

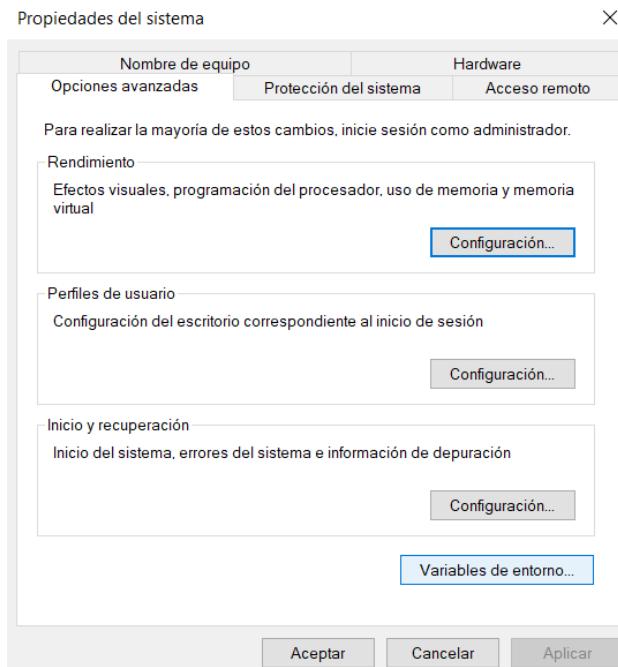


Figura 2.2: Configuración de variable de entorno en Windows.

Rutas recomendadas a añadir al PATH en Windows:

- %USERPROFILE%\dev\flutter\bin
- %ANDROID_HOME%\platform-tools
- %ANDROID_HOME%\cmdline-tools\latest\bin

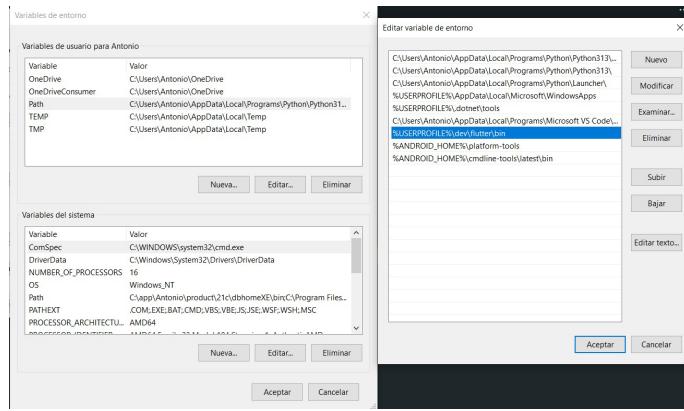


Figura 2.3: Variables de entorno a añadir.

Para verificar la instalación, ejecutar en la terminal:

```
flutter doctor
```

En Android Studio, comprobar en *File → Settings → Languages & Frameworks → Flutter* que el path está bien configurado.

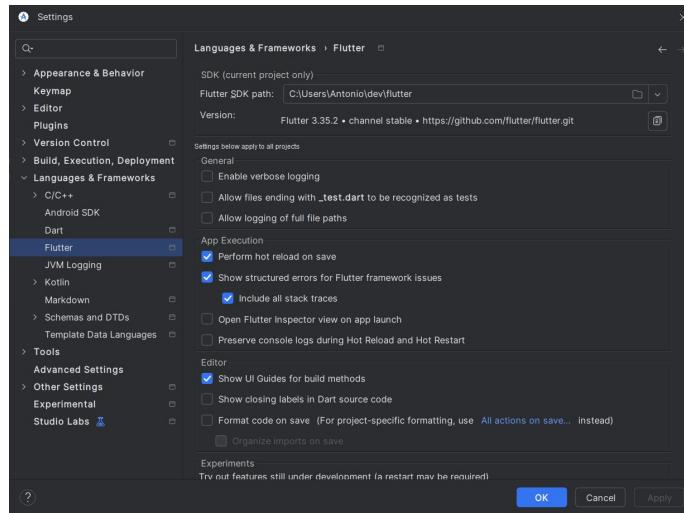


Figura 2.4: Configuración flutter en android studio.

2.6. Instalación de compiladores de C++ (Visual Studio Build Tools)

Para instalar librerías como InsightFace en el servidor FastAPI, es necesario contar con compiladores de C++.

1. Descargar desde: <https://visualstudio.microsoft.com/es/visual-cpp-build-tools/>
2. Abrir el instalador.
3. En la pestaña *Cargas de trabajo*, marcar las opciones de compilación de C++.
4. Pulsar en *Instalar*.

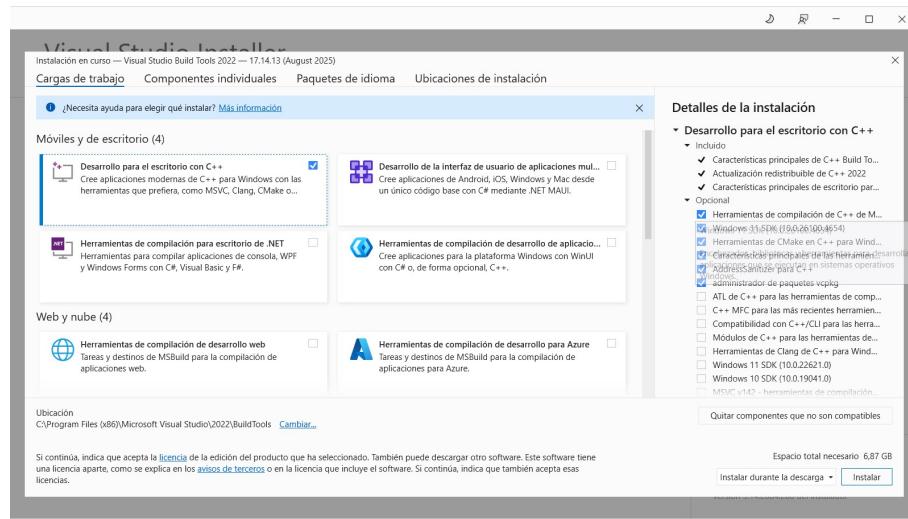


Figura 2.5: Componentes necesarios de instalación de vs build tools.

Capítulo 3

Instalación del sistema

El sistema se compone de tres elementos principales: el servidor central, el dispositivo ESP32-CAM y la aplicación móvil desarrollada en Flutter. En este capítulo se describen los pasos necesarios para instalar y configurar cada componente, así como la asignación de direcciones IP para la comunicación en red.

3.1. Instalación del servidor

El servidor es el núcleo del sistema. Para prepararlo, abre **Visual Studio Code** y selecciona la carpeta del proyecto desde: *File → Open Folder....*

Antes de ejecutarlo por primera vez, es necesario instalar las dependencias de Python usando la terminal integrada de VS Code (*View → Terminal*):

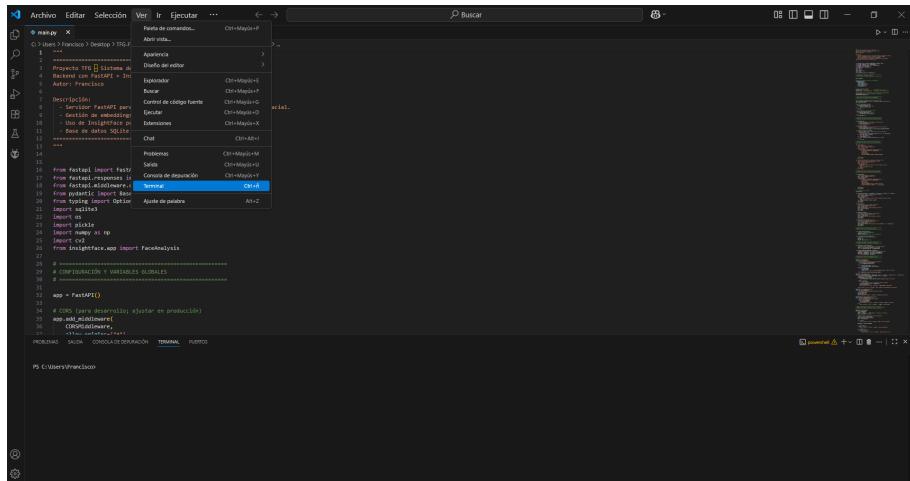


Figura 3.1: Apertura de la terminal integrada en Visual Studio Code.

```
pip install fastapi "uvicorn[standard]" htxp insightface sqlite3  
↪ python-multipart
```

Este paso se hace solo la primera vez. Con las librerías instaladas, inicia el servidor con el siguiente comando:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

Al ejecutar el comando de inicio, el servidor comienza a escuchar en el puerto 8000 y descarga los modelos de reconocimiento facial necesarios (Figura 4.3).



```
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [4744] using StatReload
download path: C:\Users\Antonio\.insightface\models\buffalo_1
Downloading C:\Users\Antonio\.insightface\models\buffalo_1.zip from https://github.com/deepinsight/insightface/releases/download/v0.7/buffalo_1.zip...
100% | 281857 / 281857 [00:41<00:00, 6840.69KB/s]
```

Figura 3.2: Ejecución del servidor FastAPI con Uvicorn e inicio de descarga de modelos de InsightFace.

Para comprobar que funciona, abre un navegador en la dirección:
<http://localhost:8000/docs>

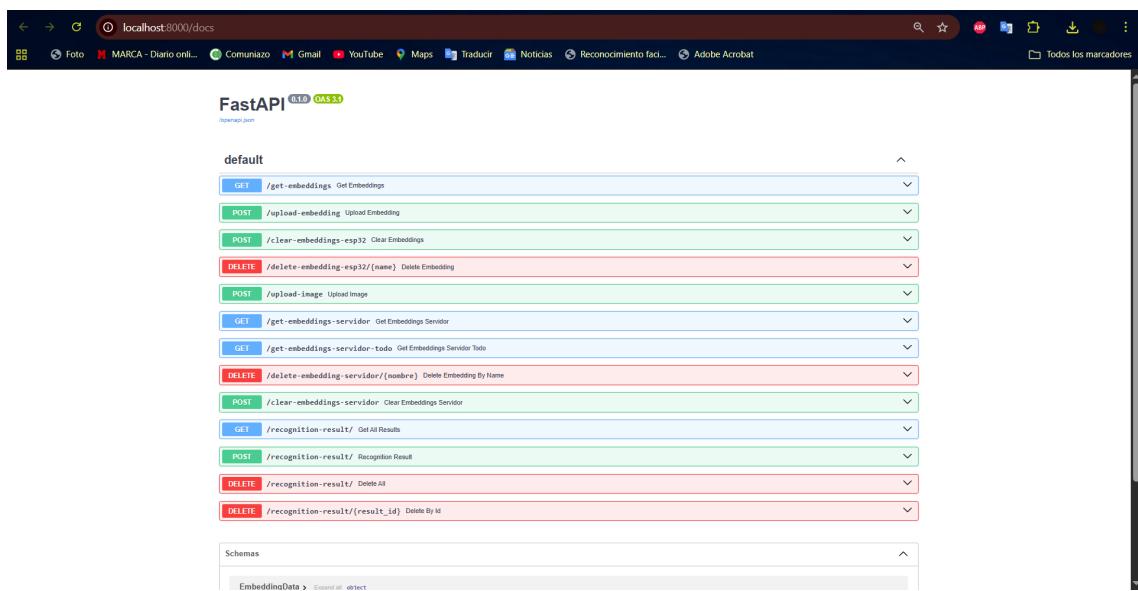


Figura 3.3: Interfaz de documentación de FastAPI al iniciar el servidor.

3.2. Instalación del dispositivo ESP32-CAM

Configuración inicial en Arduino IDE

En el menú principal de Arduino IDE selecciona: *Archivo → Preferencias*.

Como se muestra en la Figura 3.4, en el menú de preferencias de Arduino IDE se debe añadir la siguiente URL para descargar las placas ESP32.

```
https://dl.espressif.com/dl/package_esp32_index.json
```

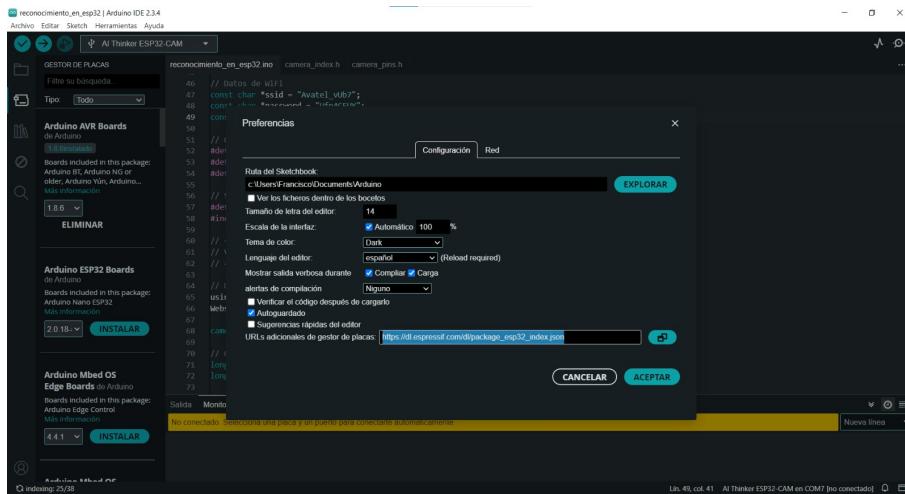


Figura 3.4: Configuración de Arduino IDE: adición de la URL de las placas ESP32 en las preferencias.

Instalación de la placa ESP32

Accede a *Herramientas → Placa → Gestor de placas*, busca ESP32 y selecciona **ESP32 by Espressif Systems**. En este proyecto se recomienda la versión **1.0.4** por compatibilidad con la librería de reconocimiento facial.

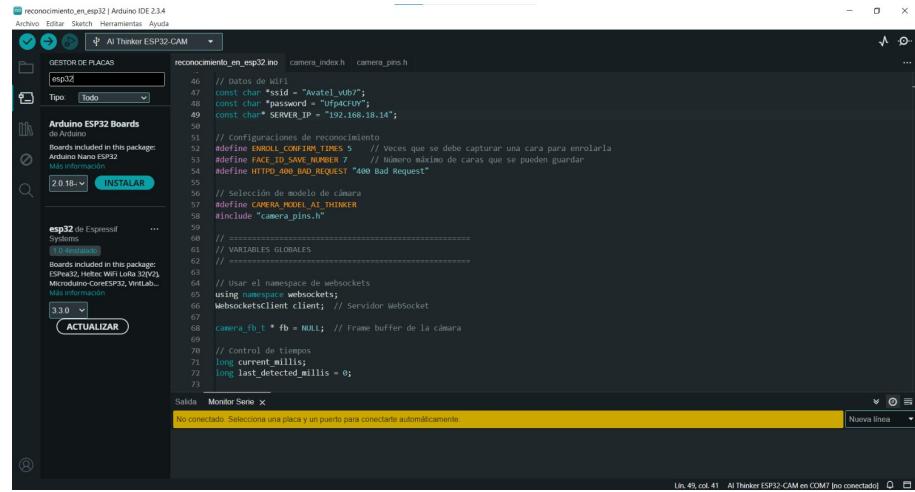


Figura 3.5: Gestor de placas en Arduino IDE mostrando el paquete ESP32 de Espressif Systems.

Instalación de librerías necesarias

Desde *Herramientas* → *Administrador bibliotecas...*, instala las siguientes:

- **ArduinoJson** (versión 7.4.2 recomendada).
- **ArduinoWebsockets** (versión 0.4.10 recomendada).
- **base64** (versión 1.3.0 recomendada).

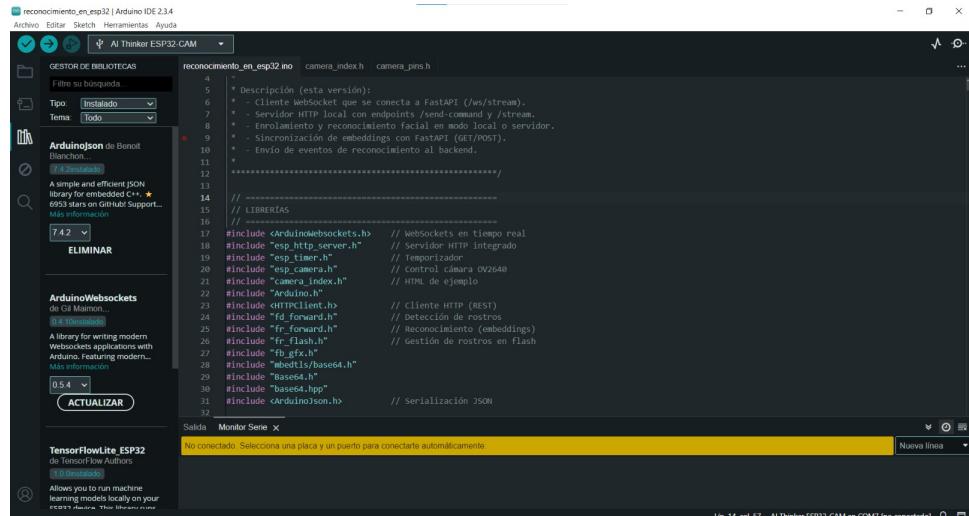


Figura 3.6: Gestor de librerías en Arduino IDE mostrando la instalación de librerías necesarias para el proyecto ESP32-CAM.

Selección de placa y puerto

Selecciona en *Herramientas* → *Placa* → *ESP32 Arduino* la opción **AI Thinker ESP32-CAM**. En *Herramientas* → *Puerto* elige el puerto COM correspondiente.

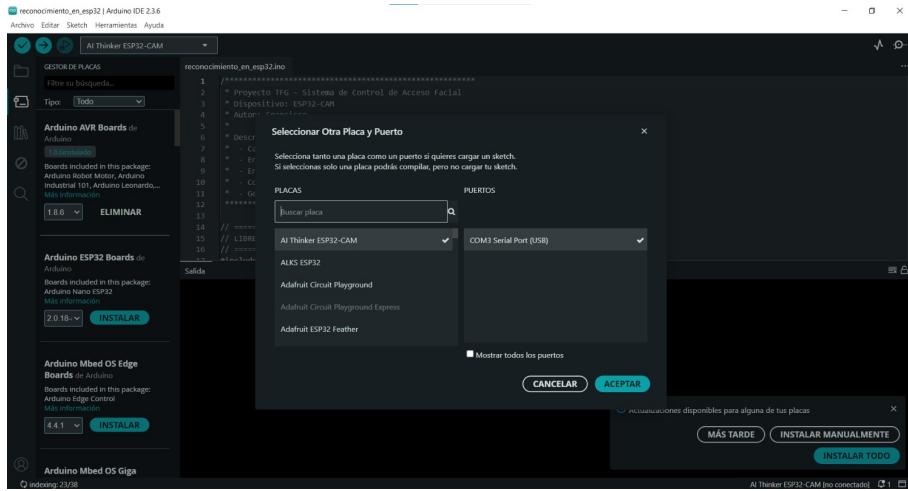


Figura 3.7: Selección de la placa AI Thinker ESP32-CAM y el puerto COM en Arduino IDE.

Configura el *Monitor Serie* a **115200 baudios**.

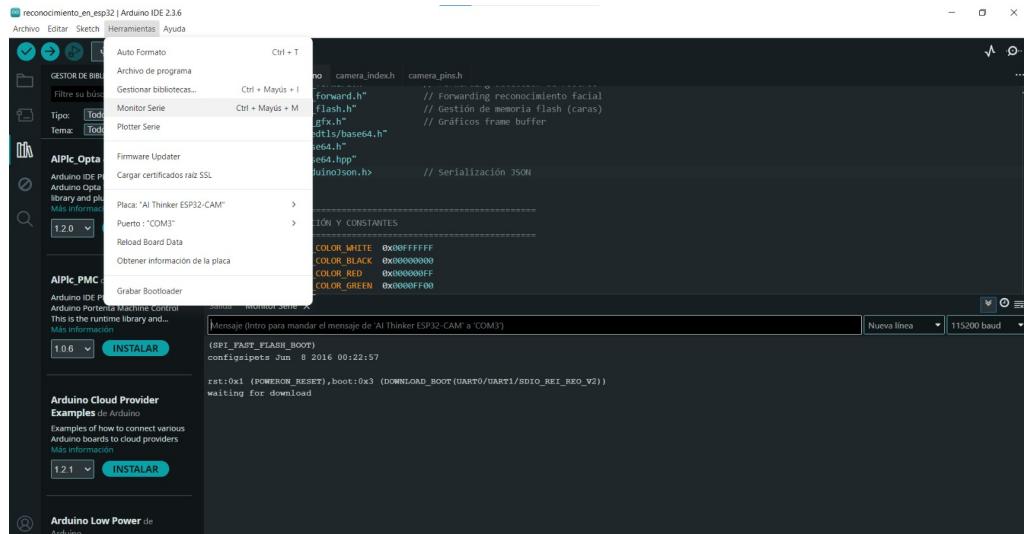


Figura 3.8: Monitor Serie en Arduino IDE configurado a 115200 baudios.

3.3. Instalación de la aplicación móvil

Abre Android Studio y selecciona *Open an Existing Project*. Elige la carpeta del proyecto Flutter.

Después, en la terminal de Android Studio ejecuta:

```
flutter pub get
```

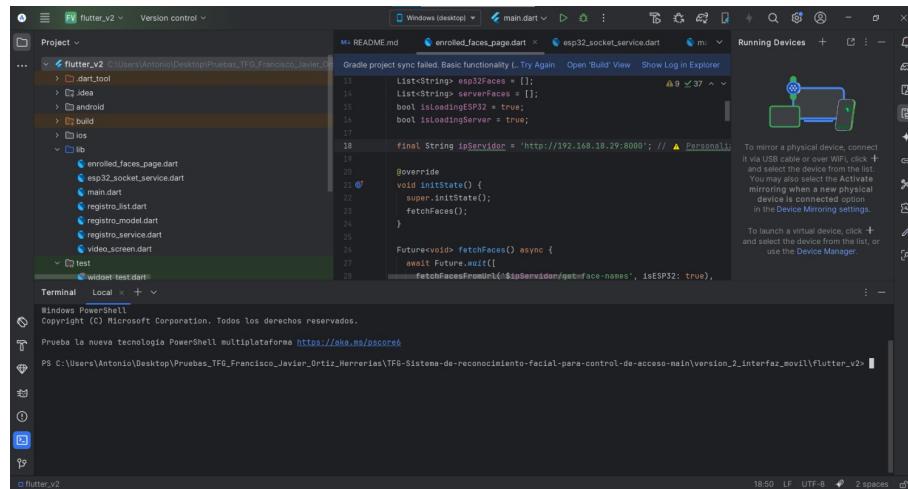


Figura 3.9: Terminal de Android Studio.

En la barra superior selecciona un emulador o un dispositivo físico conectado por USB.

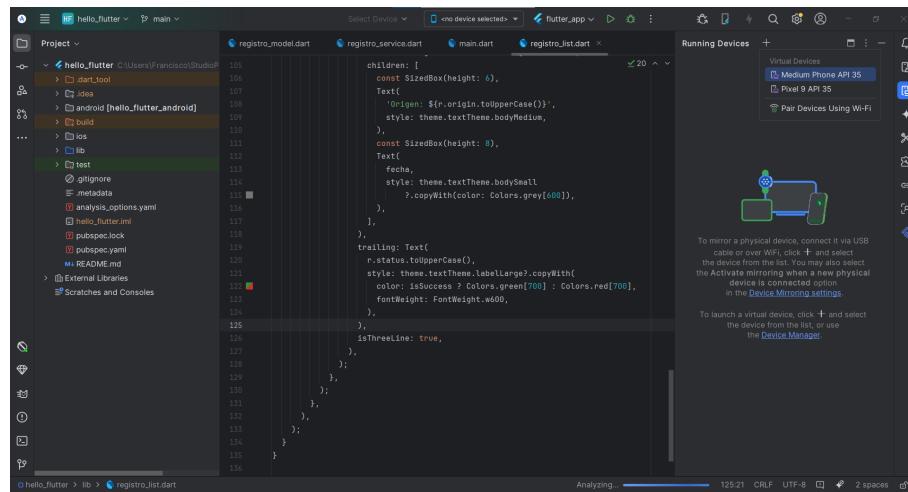


Figura 3.10: Selección de un dispositivo virtual (emulador) en Android Studio para ejecutar la aplicación Flutter.

Aparecerá el emulador de la siguiente manera:

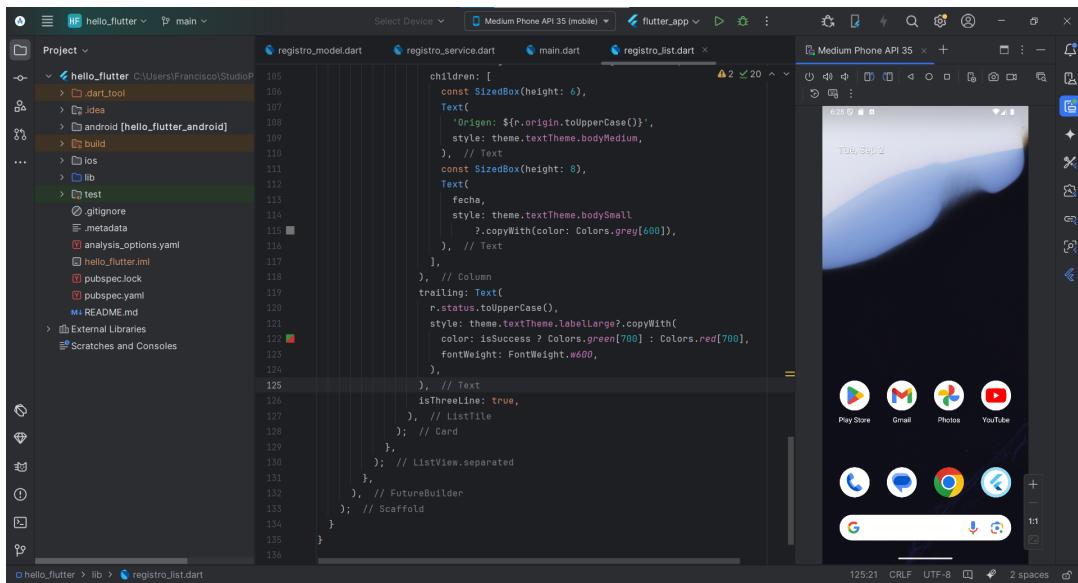


Figura 3.11: Simulador Android encendido en Android Studio, listo para ejecutar la aplicación Flutter.

Con un clic en el botón verde **Run**, la aplicación se compila e instala.

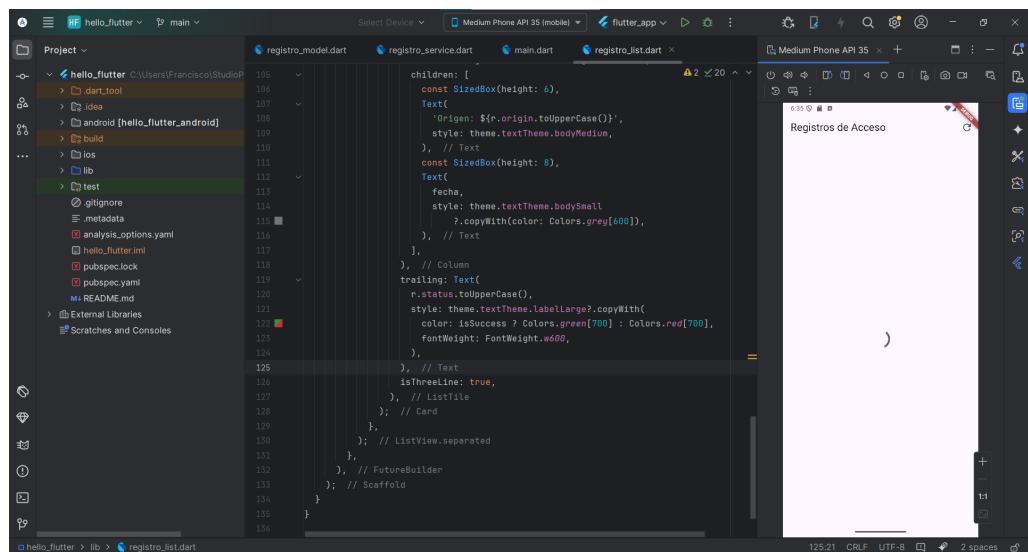


Figura 3.12: Ejecución de la aplicación Flutter en Android Studio.

3.4. Configuración de red y direcciones IP

Para que el sistema funcione correctamente, es necesario configurar credenciales WiFi y direcciones IP en el ESP32-CAM, el servidor y la aplicación móvil.

ESP32-CAM

En el código fuente modifica las credenciales WiFi y la IP del servidor:

```
const char *ssid = "NOMBRE_DE_TU_RED";
const char *password = "CONTRASEÑA_DE_TU_RED";
const char* SERVER_IP = "DIRECCION_IP_DEL_SERVIDOR";
```

Servidor FastAPI

En el archivo de configuración del servidor, asigna la IP del ESP32-CAM:

```
ESP32_IP = "192.168.1.50"
```

Aplicación móvil (Flutter)

En los archivos de la app (ejemplo `Video_screen.dart`) modifica la IP del servidor:

```
final uri = Uri.parse('http://192.168.1.100:8000/send-command');
```

Capítulo 4

Puesta en marcha

Una vez instalado y configurado cada componente, se puede proceder a la puesta en marcha del sistema. Este proceso consta de tres pasos: encender el ESP32-CAM, iniciar el servidor FastAPI y conectar la aplicación móvil.

4.1. Encendido del ESP32-CAM

Al encender el ESP32-CAM se debe verificar en el *Monitor Serie* que aparecen los siguientes mensajes:

Versión con interfaz web

En esta versión, el dispositivo muestra un mensaje indicando la dirección local a la que debe acceder el usuario para abrir la interfaz en un navegador.

```
19:38:05.124 ->
19:38:05.985 -> .....
19:38:11.474 -> WiFi connected
19:38:11.474 -> httpd_start
19:38:12.841 -> Embeddings recibidos:
19:38:12.841 -> {"pepe": [0.869336, 0.92207, -0.269141, -0.605469, -0.40175
19:38:13.824 -> Camera Ready! Use 'http://192.168.18.16' to connect
```

Figura 4.1: Mensaje mostrado por el ESP32-CAM al iniciar la versión con interfaz web.

Versión con interfaz móvil

En la versión que utiliza el servidor central, los mensajes indican la conexión WiFi y la recepción de embeddings, confirmando que el ESP32-CAM está enviando datos correctamente al backend.

```

17:33:15.660 ->
17:33:16.514 -> .....
17:33:24.515 -> WiFi connected
17:33:24.515 -> IP ESP32: 'http://192.168.18.16
17:33:24.754 -> Conectado a FastAPI WS
17:33:24.754 -> Conectando WS a ws://192.168.18.14:8000/ws/stream
17:33:24.790 -> HTTP Server started, handlers registered.
17:33:25.031 -> Embeddings recibidos:
17:33:25.031 -> {}
17:33:25.031 -> Intentando conectar WS...
17:33:25.064 -> ● Cliente WebSocket conectado

```

Figura 4.2: Mensajes del ESP32-CAM indicando conexión con el servidor FastAPI.

Si la conexión es correcta, el servidor muestra un mensaje confirmando la comunicación.

4.2. Inicio del servidor FastAPI

Para ejecutar el servidor, en Visual Studio Code se debe lanzar el siguiente comando:

```
uvicorn main:app --reload --host 0.0.0.0 --port 8000
```

Si todo se realizó correctamente, la terminal mostrará un resultado similar al de la Figura 4.3, confirmando que el servidor está escuchando en el puerto 8000.

```

PS C:\Users\Francisco\Desktop\TFG-Francisco\version 1.interfaz web\servidor> uvicorn main:app --host 0.0.0.0 --port 8000
Applied providers: ['CPUExecutionProvider'], with options: {'CPUExecutionProvider': {}}
find model: C:\Users\Francisco/.insightface\models\buffalo_1\1k3d68.onnx landmark_3d_68 ['None', 3, 192, 192] 0.0 1.0
Applied providers: ['CPUExecutionProvider'], with options: {'CPUExecutionProvider': {}}
find model: C:\Users\Francisco/.insightface\models\buffalo_1\2d106det.onnx landmark_2d_106 ['None', 3, 192, 192] 0.0 1.0
Applied providers: ['CPUExecutionProvider'], with options: {'CPUExecutionProvider': {}}
find model: C:\Users\Francisco/.insightface\models\buffalo_1\det_10g.onnx detection [1, 3, '?', '?'] 127.5 128.0
Applied providers: ['CPUExecutionProvider'], with options: {'CPUExecutionProvider': {}}
find model: C:\Users\Francisco/.insightface\models\buffalo_1\genderage.onnx genderage ['None', 3, 96, 96] 0.0 1.0
Applied providers: ['CPUExecutionProvider'], with options: {'CPUExecutionProvider': {}}
find model: C:\Users\Francisco/.insightface\models\buffalo_1\w600k_r50.onnx recognition ['None', 3, 112, 112] 127.5 127.5
set det-size: (640, 640)
INFO:     Started server process [15808]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

```

Figura 4.3: Ejecución del servidor FastAPI en Visual Studio Code.

Cuando la conexión con las interfaces es exitosa, en la terminal aparecen registros de las peticiones recibidas.

```
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 127.0.0.1:49954 - "GET /get-embeddings-servidor HTTP/1.1" 200 OK
INFO: 127.0.0.1:52428 - "GET /get-embeddings-servidor HTTP/1.1" 200 OK
INFO: 127.0.0.1:52428 - "GET /get-embeddings-servidor HTTP/1.1" 200 OK
INFO: 127.0.0.1:52429 - "GET /get-embeddings-servidor HTTP/1.1" 200 OK
```

Figura 4.4: Conexión de la interfaz con el servidor FastAPI.

4.3. Conexión de la aplicación móvil

Al abrir la aplicación móvil en Android Studio, si la conexión con el servidor es exitosa, en la terminal de Visual Studio Code se mostrarán peticiones similares a las siguientes:

```
set det-size: (640, 640)
INFO: Started server process [17484]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: ('192.168.18.16', 63706) - "WebSocket /ws/stream" [accepted]
Cliente conectado: 192.168.18.16
INFO: connection open
Conexión cerrada o error: Cannot call "receive" once a disconnect message has been received.
INFO: connection closed
INFO: ('192.168.18.16', 54229) - "WebSocket /ws/stream" [accepted]
Cliente conectado: 192.168.18.16
INFO: connection open
INFO: 192.168.18.16:60484 - "GET /get-embeddings HTTP/1.1" 200 OK
Conexión cerrada o error: Cannot call "receive" once a disconnect message has been received.
INFO: connection closed
INFO: ('192.168.18.16', 65251) - "WebSocket /ws/stream" [accepted]
Cliente conectado: 192.168.18.16
INFO: connection open
Mensaje de texto recibido del ESP32: delete_faces
INFO: 192.168.18.14:50301 - "GET /recognition-result/ HTTP/1.1" 200 OK
```

Figura 4.5: Conexión de la aplicación Flutter con el servidor FastAPI.

Capítulo 5

Funcionamiento del sistema

Una vez instalado y puesto en marcha, el sistema puede utilizarse a través de dos interfaces diferentes:

- **Interfaz web**, accesible desde un navegador en el ordenador.
- **Aplicación móvil Flutter**, que ofrece una versión adaptada para dispositivos Android.

Ambas interfaces permiten gestionar usuarios, realizar reconocimientos y consultar accesos, aunque la aplicación móvil está más orientada al uso en movilidad, mientras que la interfaz web concentra todas las funciones de administración.

5.1. Interfaz web del sistema

Al cargar el firmware del ESP32-CAM y abrir el Monitor Serie de Arduino IDE, se mostrará un mensaje confirmando la conexión a la red WiFi. En este mensaje también aparece un enlace con la dirección IP asignada al dispositivo. Dicho enlace corresponde a la interfaz web del sistema y permite acceder desde cualquier navegador de la misma red local.

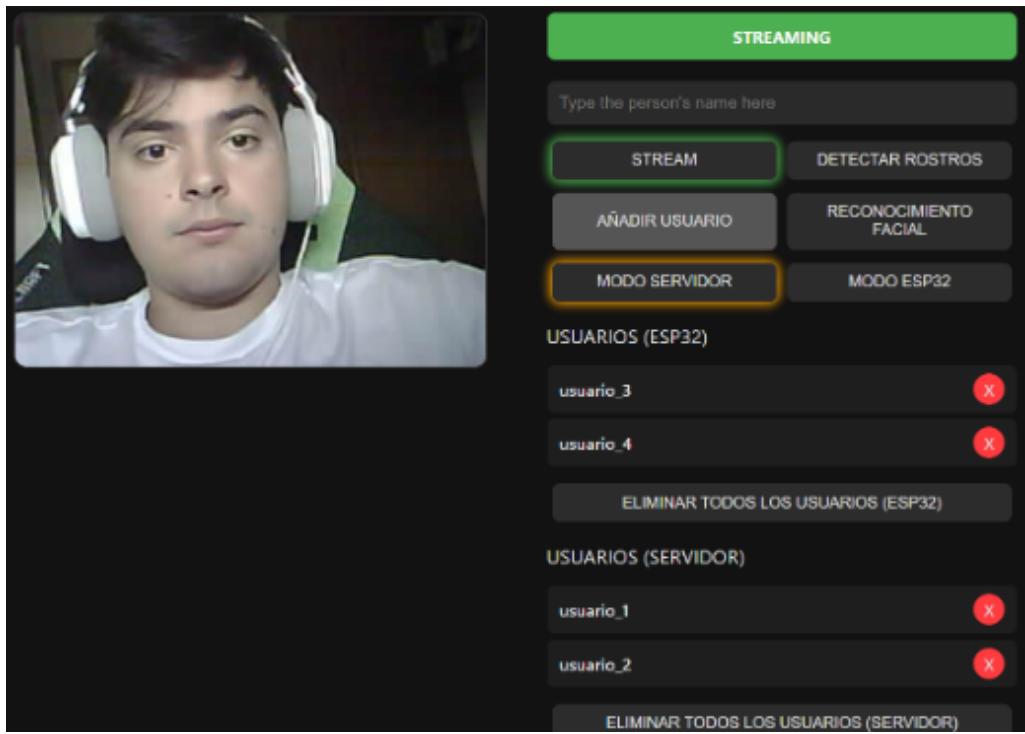


Figura 5.1: Interfaz web del sistema de control de acceso.

La interfaz web cuenta con dos secciones principales:

- **Zona de vídeo (izquierda)**: muestra la transmisión en directo de la cámara del ESP32-CAM. Sirve para comprobar en tiempo real la posición de la persona frente a la cámara.
- **Panel de control (derecha)**: compuesto por botones y listas que permiten gestionar el sistema.

Dentro del panel de control se encuentran:

Barra de estado (parte superior)

Una barra de color verde indica dinámicamente el estado del sistema:

- “STREAMING” cuando se mantiene la transmisión de vídeo.
- Mensajes como “Detectando rostros...” o “Captura 1 de 5” durante operaciones específicas.
- Bienvenida con el nombre del usuario reconocido o aviso de acceso denegado.

Botones principales

- **STREAM:** inicia o detiene la transmisión.
- **DETECTAR ROSTROS:** analiza la imagen para localizar caras.
- **AÑADIR USUARIO:** registra un nuevo usuario.
- **RECONOCIMIENTO FACIAL:** compara caras detectadas con la base de datos.
- **MODO SERVIDOR / MODO ESP32:** cambia el modo de procesamiento.

Gestión de usuarios

- **Usuarios (ESP32):** muestra los usuarios registrados mediante el modo local del ESP32-CAM.
- **Usuarios (Servidor):** muestra los usuarios registrados mediante el modo remoto a través del servidor FastAPI.

Eliminación masiva

- **Eliminar todos los usuarios (ESP32):** elimina del backend todos los usuarios registrados en modo local (ESP32).
- **Eliminar todos los usuarios (Servidor):** elimina del backend todos los usuarios registrados en modo remoto (servidor FastAPI).

5.2. Interfaz móvil

La aplicación móvil en Flutter permite manejar el sistema de manera sencilla. Cuenta con tres pantallas principales:

5.2.1. Historial de accesos

Muestra todos los intentos de acceso, exitosos o fallidos, indicando nombre del usuario, fecha, hora, origen de la verificación (ESP32 o Servidor) y resultado (Figura 5.2). Los accesos correctos aparecen en verde (**SUCCESS**), mientras que los fallidos aparecen en rojo (**ERROR**).

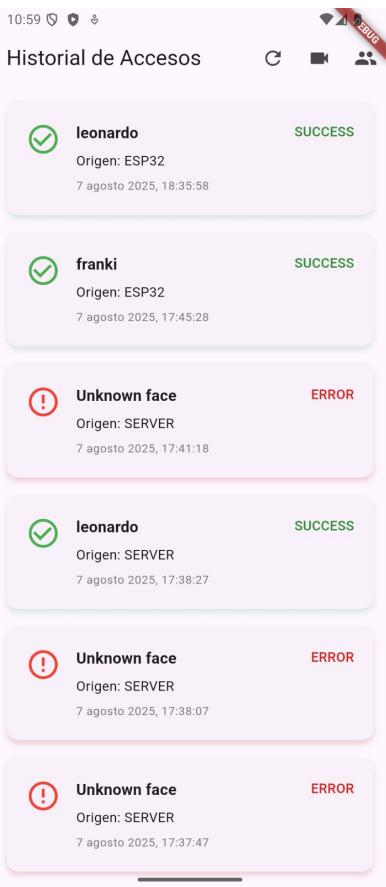


Figura 5.2: Pantalla de historial de accesos en la aplicación Flutter.

5.2.2. Cámara en vivo

Muestra la transmisión en directo del ESP32-CAM, indicando en la parte superior el modo activo y el tipo de procesamiento. Los carteles amarillos informan en tiempo real sobre el estado, progreso y resultados de las acciones (similar a la barra de estado de la interfaz web (Figura 5.3).

Debajo de estos mensajes se encuentran los botones de acción:

- Iniciar Stream.
- Iniciar Detección.
- Reconocer Rostro.
- Modo ESP32 / Modo Servidor.

Finalmente, en la parte inferior hay un campo de texto para introducir el nombre de un usuario y un botón **Enrolar Rostro**, que inicia el registro de una nueva persona.

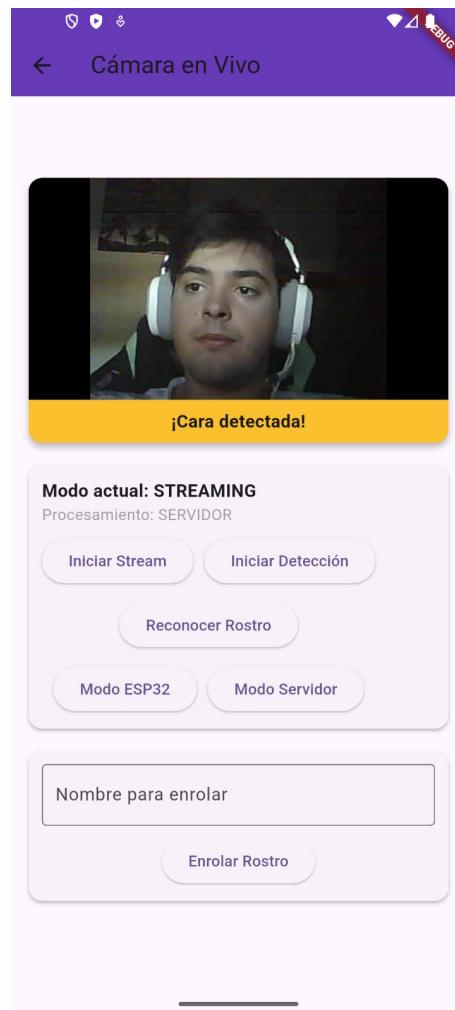


Figura 5.3: Pantalla de cámara en vivo en la aplicación Flutter.

5.2.3. Rostros enrolados

Esta pantalla muestra la lista de usuarios registrados en el sistema. En la parte superior se puede alternar entre ESP32 y Servidor para visualizar dónde están almacenados los rostros.

- Cada usuario aparece con un botón rojo para eliminarlo individualmente.
- En la parte inferior existe un botón para eliminar todos los registros de golpe.



Figura 5.4: Pantalla de usuarios registrados en la aplicación Flutter.

Capítulo 6

Resolución de problemas comunes

En esta sección se describen los errores más habituales durante la instalación o el uso del sistema, junto con sus posibles causas y soluciones.

6.1. Emulador de Android “congelado” o con imagen deformada

Síntomas: El emulador se queda bloqueado, muestra la pantalla distorsionada o no arranca la aplicación. (Es el mismo fallo que ilustran las dos imágenes.)

Causa habitual: Corrupción de datos del AVD/snapshot o configuración de dispositivo virtual incompatible.

Solución rápida:

1. Abrir *Device Manager* en Android Studio.
2. En el dispositivo afectado, abrir el menú y pulsar *Wipe Data*.
3. Si sigue fallando, eliminar el AVD y crear uno nuevo (*Add Device* → *Phone* → *perfil Pixel recomendado* → *Finish*).

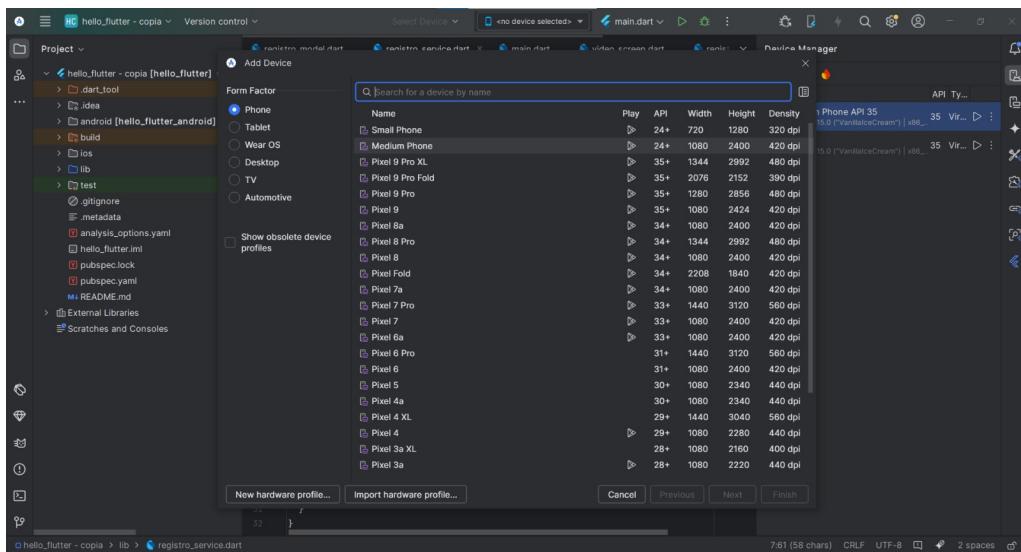


Figura 6.1: Selección de un dispositivo virtual en Android Studio para crear o reiniciar un emulador.

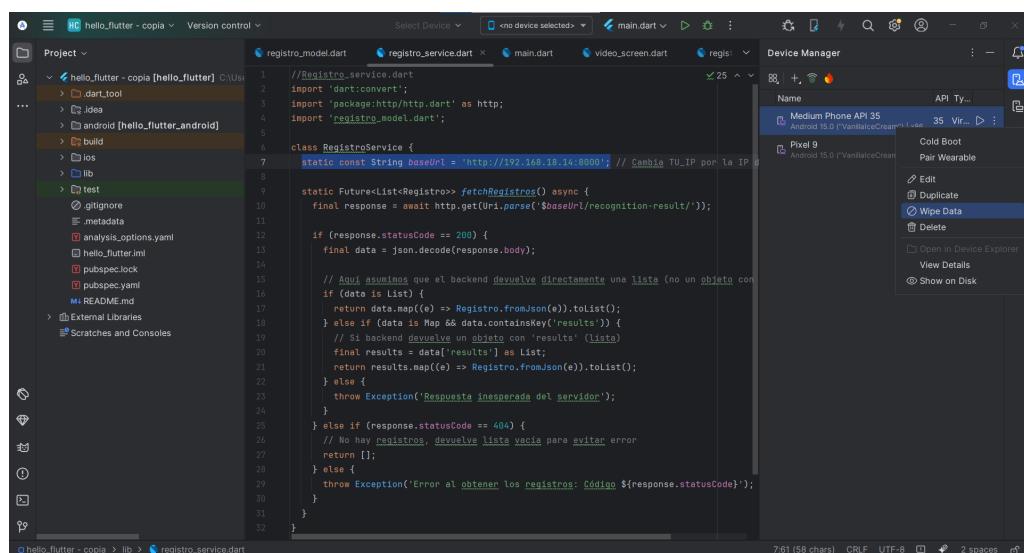
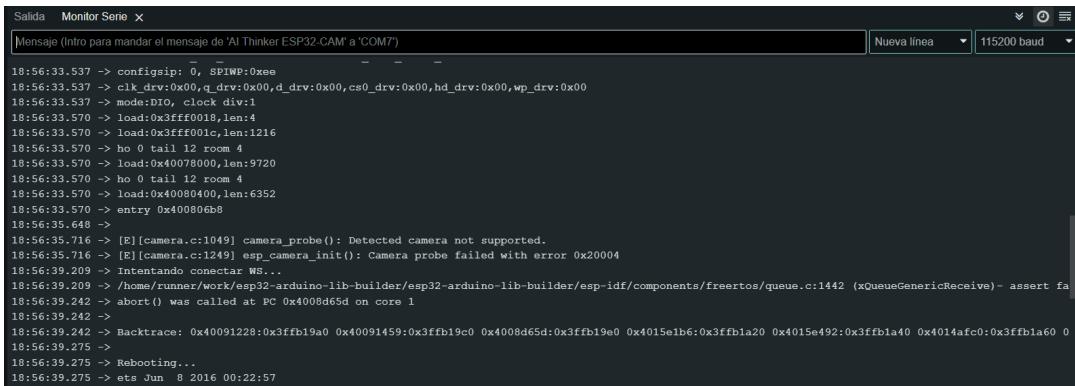


Figura 6.2: Opción *Wipe Data* en el menú del emulador desde el Device Manager de Android Studio.

6.2. Error con la cámara del ESP32-CAM

Al abrir el Monitor Serie después de cargar el firmware, el dispositivo puede mostrar este error.



The screenshot shows the Arduino Serial Monitor window titled "Monitor Serie". The text area contains the following log output:

```
Salida  Monitor Serie x
Mensaje (Intro para mandar el mensaje de 'AI Thinker ESP32-CAM' a 'COM7')
18:56:33.537 -> configisp: 0, SPIWP:0xee
18:56:33.537 -> clk_drv:0x00,q_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
18:56:33.537 -> mode:DIO, clock div:1
18:56:33.570 -> load:0x3fff0018,len:4
18:56:33.570 -> load:0x3fff001c,len:1216
18:56:33.570 -> ho 0 tail 12 room 4
18:56:33.570 -> load:0x40078000,len:9720
18:56:33.570 -> ho 0 tail 12 room 4
18:56:33.570 -> load:0x40080400,len:6352
18:56:33.570 -> entry 0x40080668
18:56:35.648 ->
18:56:35.716 -> [E] [camera.c:1049] camera_probe(): Detected camera not supported.
18:56:35.716 -> [E] [camera.c:1249] esp_camera_init(): Camera probe failed with error 0x20004
18:56:39.209 -> Intentando conectar WS...
18:56:39.209 -> /home/runner/work/esp32-arduino-lib-builder/esp32-arduino-lib-builder/esp-idf/components/freertos/queue.c:1442 (xQueueGenericReceive) - assert failed
18:56:39.242 -> abort() was called at PC 0x4008d65d on core 1
18:56:39.242 ->
18:56:39.242 -> Backtrace: 0x40091228:0x3ffb19a0 0x40091459:0x3ffb19c0 0x4008d65d:0x3ffb19e0 0x4015e1b6:0x3ffb1a20 0x4015e492:0x3ffb1a40 0x4014afc0:0x3ffb1a60 0
18:56:39.275 ->
18:56:39.275 -> Rebooting...
18:56:39.275 -> ets Jun 8 2016 00:22:57
```

Figura 6.3: Error en el Monitor Serie: `camera_probe(): Detected camera not supported.`

Este mensaje suele aparecer tras un reinicio del dispositivo. No siempre implica un fallo de hardware, sino que muchas veces ocurre de forma aleatoria al volver a arrancar la cámara.

Solución recomendada: Pulsar el botón de reset situado en la parte trasera del ESP32-CAM o desconectarlo y volverlo a conectar por USB. En la mayoría de los casos, esto soluciona el problema.

6.3. ESP32-CAM no conectado en Arduino IDE

En la parte inferior del entorno de Arduino IDE puede aparecer un mensaje en amarillo indicando que el dispositivo no está conectado.

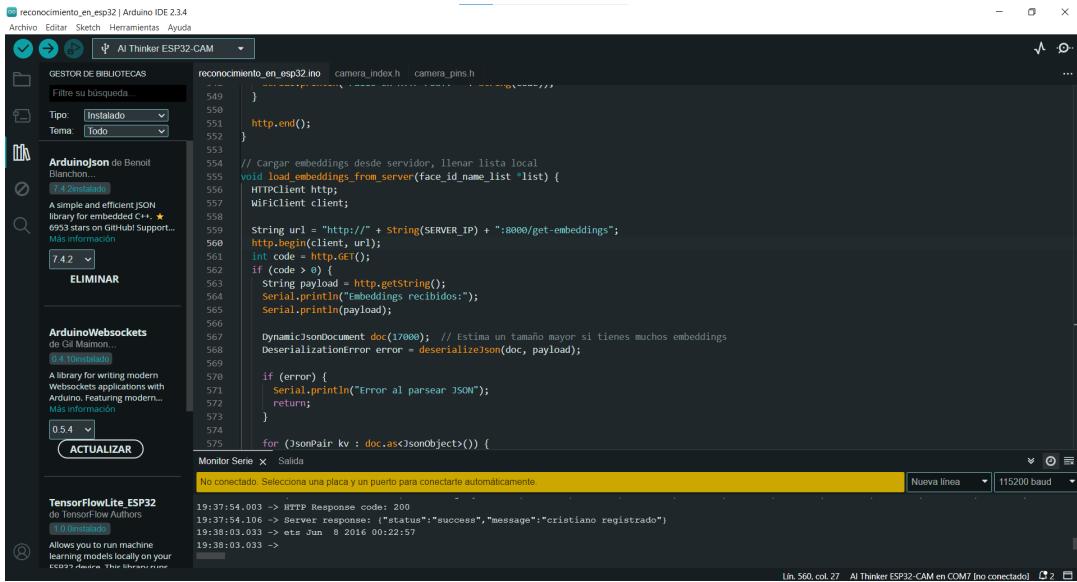


Figura 6.4: Aviso en Arduino IDE indicando que el ESP32-CAM no está conectado.

Causas posibles:

- No se ha seleccionado la placa correcta.
- No se ha elegido un puerto COM.
- El cable USB no admite transmisión de datos (solo carga).

Soluciones:

1. Ir a *Herramientas* → *Placa* y seleccionar **AI Thinker ESP32-CAM**.
2. En *Herramientas* → *Puerto*, elegir el COM correspondiente.
3. Probar con un cable USB que sí soporte datos.
4. Si sigue sin aparecer, reinstalar los drivers (CH340 o CP2102, según el módulo).

6.4. La aplicación se queda cargando en el historial de accesos

En ocasiones, al abrir la sección *Historial de Accesos*, la pantalla muestra un círculo de carga que no desaparece. Esto ocurre cuando la aplicación móvil no consigue establecer comunicación con el servidor FastAPI.

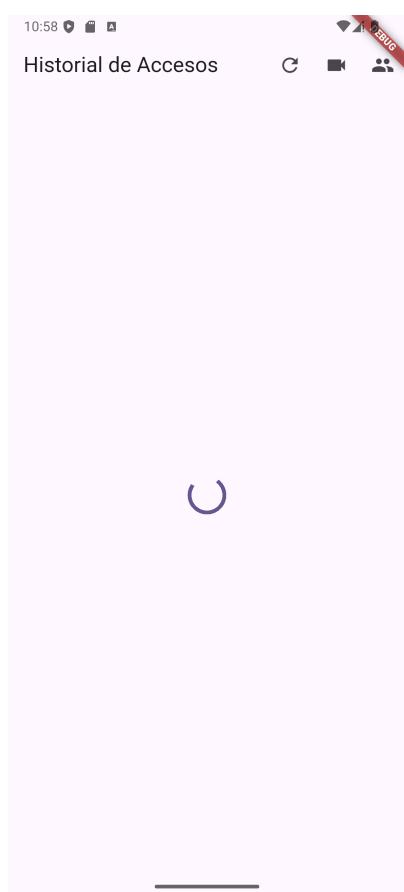


Figura 6.5: La aplicación móvil queda en estado de carga al abrir el historial de accesos.

Soluciones:

1. Comprobar las direcciones IP configuradas en la app (ver apartado 2.4).
2. Verificar que el servidor FastAPI está en ejecución en el puerto 8000.
3. Confirmar que el dispositivo móvil está conectado a la misma red WiFi que el servidor.

6.5. Monitor Serie muestra caracteres extraños

En el Monitor Serie aparecen caracteres ilegibles al ejecutar el firmware. Esto sucede porque la velocidad de comunicación (*baudios*) configurada en Arduino IDE no coincide con la del programa.

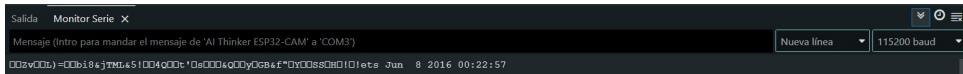


Figura 6.6: Monitor Serie mostrando caracteres ilegibles por tener configurada una velocidad incorrecta.

Solución: Seleccionar **115200 baudios** en la esquina inferior derecha del Monitor Serie. Después, resetear el dispositivo para que los mensajes se muestren correctamente.

6.6. Problemas durante la instalación de dependencias

Al instalar las librerías necesarias para el servidor puede aparecer el error:
Microsoft Visual C++ 14.0 or greater is required

```

PROBLEMS 7 DEBUG CONSOLE TERMINAL PORTS

!! check.warn(importable)
creating build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core.cpp -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core.h -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core_cython.cpp -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
creating build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\Tom_Hanks_54745.png -> build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\mask_black.jpg -> build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\mask_blue.jpg -> build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\mask_green.jpg -> build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\mask_white.jpg -> build\lib.win-amd64-cpython-313\insightface\data\images
copying insightface\data\images\t1.jpg -> build\lib.win-amd64-cpython-313\insightface\data\images
creating build\lib.win-amd64-cpython-313\insightface\data\objects
copying insightface\data\objects\meanshape_68.pkl -> build\lib.win-amd64-cpython-313\insightface\data\objects
copying insightface\thirdparty\face3d\mesh\cython\mesh_core_cython.c -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core_cython.cpp -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core_cython.h -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\mesh_core_cython.pyc -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
copying insightface\thirdparty\face3d\mesh\cython\setup.py -> build\lib.win-amd64-cpython-313\insightface\thirdparty\face3d\mesh\cython
running build ext
error: Microsoft Visual C++ 14.0 or greater is required. Get it with "Microsoft C++ Build Tools": https://visualstudio.microsoft.com/visual-cpp-build-tools/
[end of output]

note: This error originates from a subprocess, and is likely not a problem with pip.
ERROR: Failed building wheel for insightface
Failed to build insightface
error: failed-wheel-build-for-install

```

Figura 6.7: Error al instalar dependencias: se requiere Microsoft Visual C++ 14.0 o superior.

Causa: Algunas dependencias, como **insightface**, requieren herramientas de compilación adicionales en Windows.

Solución: Instalar **Visual Studio Build Tools** (ver apartado 2.5). Una vez instalado, volver a ejecutar el comando de instalación.

6.7. SDK de Android incompleto en Android Studio

En algunos casos, al compilar la aplicación aparece un error porque faltan herramientas del Android SDK.

Solución:

1. Abrir *File* → *Settings* → *Languages & Frameworks* → *Android SDK*.
2. En la pestaña *SDK Tools*, asegurarse de que están instalados:
 - Android SDK Build-Tools
 - Android Emulator
 - Android SDK Platform-Tools
 - Android SDK Command-line Tools (latest)

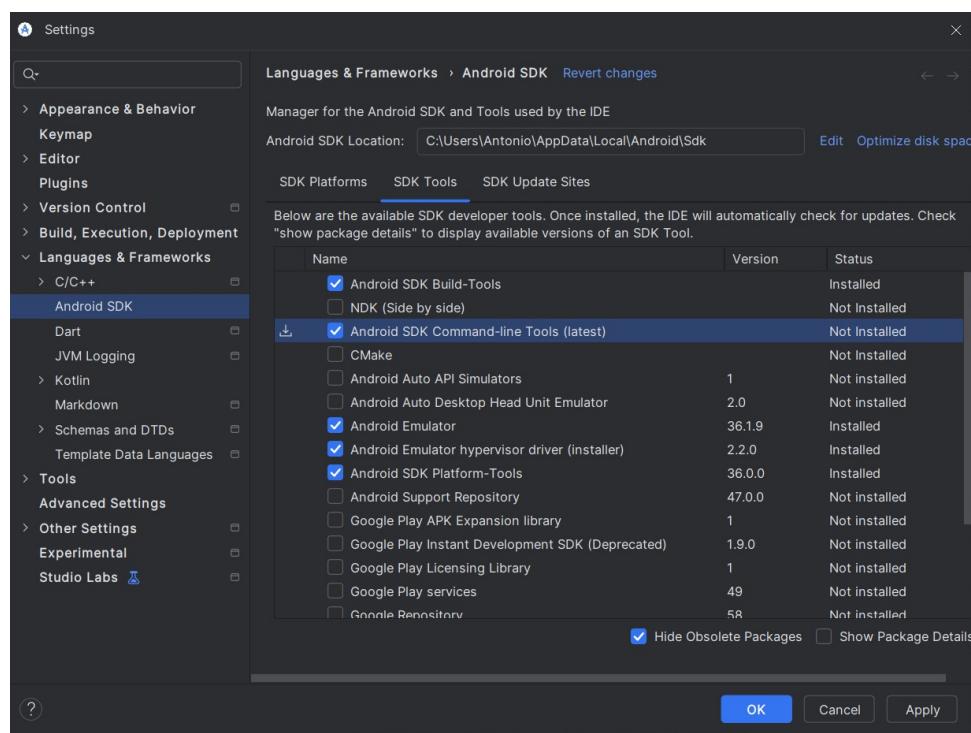


Figura 6.8: Configuración de Android Studio para instalar herramientas del Android SDK.

6.8. Error al ejecutar flutter pub get (conflicto con intl)

Puede aparecer un error de dependencias, por ejemplo con `intl`.

```
PS C:\Users\Antonio\Desktop\Pruebas_TFG_Francisco_Javier_Oritz_Herreras\TFG-Sistema-de-reconocimiento-facial-para-control-de-acceso-main\version_2_interfaz_movil\flutter_v2> flutter pub get
Resolving dependencies... (1.0s)
Note: intl is pinned to version 0.20.2 by flutter_localizations from the flutter SDK.
See https://dart.dev/guides/sdk-version-pinning for details.

Because hello_flutter depends on flutter_localizations from sdk which depends on intl 0.20.2, intl 0.20.2 is required.
So, because hello_flutter depends on intl ^0.19.0, version solving failed.

You can try the following suggestion to make the pubspec resolve:
  * Try upgrading your constraint on intl: flutter pub add intl:^0.20.2
Failed to update packages.
```

Figura 6.9: Error en la ejecución de `flutter pub get` debido a conflicto de versiones con la librería `intl`.

Solución:

```
flutter pub add intl:^0.20.2
flutter pub get
```

6.9. Dart SDK not configured

En ocasiones, al abrir el proyecto Flutter en Android Studio aparece el aviso: `Dart SDK is not configured`.

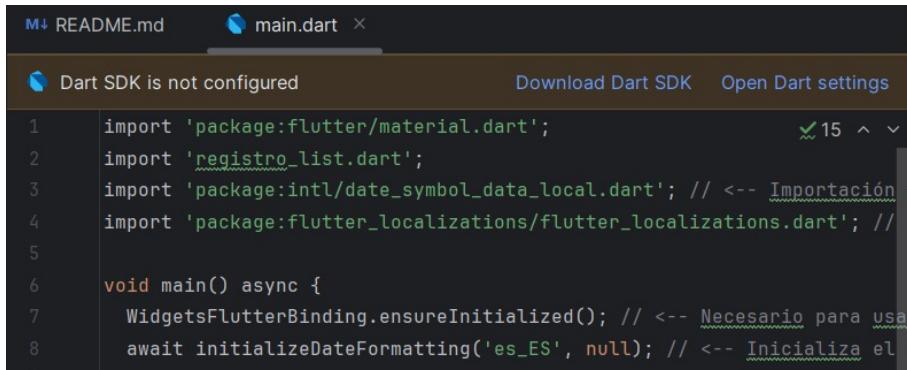


Figura 6.10: Advertencia en Android Studio indicando que el Dart SDK no está configurado.

Solución:

1. Pulsar en *Download Dart SDK*.
2. Comprobar que la ruta está correctamente asignada en: *Settings → Languages & Frameworks → Dart*.

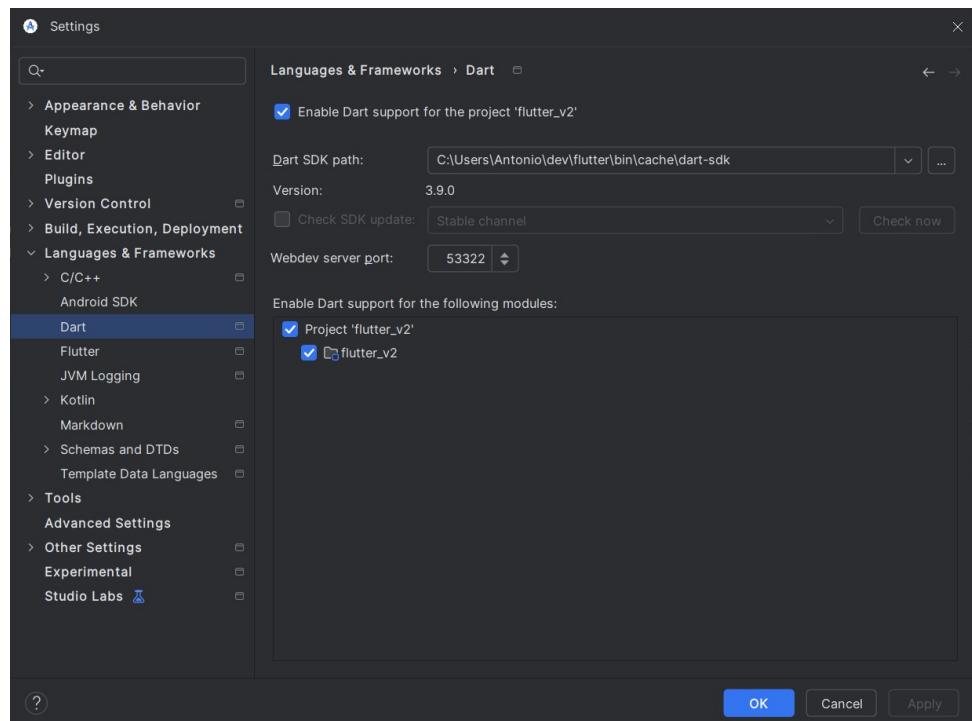


Figura 6.11: Configuración de la ruta del Dart SDK en Android Studio.

Capítulo 7

Preguntas frecuentes (FAQ)

¿Qué pasa si cierro Visual Studio Code mientras el sistema está en marcha?

El servidor FastAPI dejará de funcionar. Para volver a utilizar el sistema, será necesario reiniciarlo ejecutando el comando de **Uvicorn**.

¿Cuántos usuarios puedo registrar en el sistema? El número depende de la memoria utilizada:

- En el ESP32-CAM, el límite es reducido debido a las limitaciones de memoria. Aunque en algunas pruebas se llegó a registrar hasta 11 usuarios, la recomendación es mantenerlo entre 5 y 10 para asegurar un funcionamiento estable.
- En el servidor FastAPI, el número es mucho mayor. En las pruebas realizadas se superaron los 50 usuarios registrados sin problemas. Sin embargo, no se han realizado pruebas exhaustivas de rendimiento con cifras más altas, por lo que no puede garantizarse un límite máximo.

¿Qué diferencia hay entre el modo servidor y el modo ESP32? En modo **ESP32**, todo el procesamiento (detección y reconocimiento) se hace directamente en el microcontrolador. En **modo servidor**, el ESP32-CAM envía las imágenes al backend en FastAPI, que se encarga del reconocimiento facial con mayor precisión y capacidad.

¿El sistema funciona sin conexión a Internet? El sistema no necesita acceso a Internet global para funcionar, pero sí requiere que todos los componentes (ESP32-CAM, servidor y aplicación móvil) estén conectados a la misma red local.

Esto se debe a que la comunicación entre dispositivos se realiza mediante mensajes HTTP y WebSocket, que necesitan direcciones IP válidas dentro de la red.

En la práctica, basta con que todos los dispositivos estén conectados al mismo router WiFi. No importa si el router no tiene salida a Internet, siempre que permita la comunicación interna entre los equipos.

Para acceder desde fuera de la red local (por ejemplo, desde un móvil con datos 4G), sí sería necesario configurar el servidor con acceso público a Internet y abrir los puertos correspondientes.

¿Por qué a veces la imagen se ve trabada o con retraso? La calidad del vídeo depende de la conexión WiFi local. Si la señal es débil, hay interferencias o la red está saturada, los paquetes de datos se retrasan o se pierden, y la transmisión se ve entrecortada. Con una buena conexión WiFi (señal fuerte y baja interferencia), la transmisión es mucho más fluida.

¿Cómo borro todos los usuarios registrados? Tanto en la interfaz web como en la aplicación móvil existen botones para borrar todos los usuarios de golpe. También se pueden eliminar uno a uno mediante el botón rojo junto a cada nombre.

¿Qué hago si cambio de red WiFi? Si se cambia la red WiFi, es necesario actualizar las credenciales de conexión y las direcciones IP tanto en el ESP32-CAM como en el servidor y la aplicación móvil. Este proceso se explica con detalle en el apartado 2.4 *Configuración de red e IPs*.