UNIVERSIDAD DE CASTILLA LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Proyecto de laboratorio: Aplicación SignalScan

Multimedia

Raúl González Velázquez
Laura Fernández del Moral Gª Consuegra

Curso 2023/2024

Contenido

1. INTRODUCCIÓN

En el desarrollo del proyecto de prácticas de la asignatura de Multimedia, desarrollaremos una aplicación móvil enfocada en facilitar el reconocimiento y la comprensión de señales de tráfico mediante el aprovechamiento de las tecnologías actuales. La selección de este tema se fundamenta en su importancia crítica para la seguridad vial y la necesidad de abordar de manera innovadora y educativa un aspecto esencial de la movilidad.

1.1. DESCRIPCIÓN DEL PROYECTO

Este proyecto se basa en el desarrollo de una aplicación móvil utilizando **Visual Studio Code** y **Flutter**. La aplicación se especializa en la identificación y análisis de señales de tráfico, brindando a los usuarios una herramienta eficaz para comprender y gestionar la información vial de manera práctica y sencilla.

La aplicación incluirá una variedad de funcionalidades, la principal será permitir a los usuarios escanear señales de tráfico utilizando la cámara de su dispositivo móvil, ya sea capturando una foto en el momento o seleccionando una imagen desde la galería. Tras este proceso, la aplicación realiza un análisis detallado de la señal escaneada, proporcionando al usuario información clave y detalles relevantes sobre la misma. Además, incluiremos también un apartado de noticias, relacionadas con el tráfico y la seguridad vial, permitiendo así a los usuarios estar informados de los últimos acontecimientos ocurridos; también, incluirá otro apartado de información detallada de cada una de las señales de tráfico existentes.

La interfaz de usuario será sido diseñada para ser intuitiva y fácil de usar, garantizando que los usuarios puedan acceder a la información de manera rápida y eficiente. Elementos visuales, como imágenes representativas de las señales y detalles informativos, se incorporan de manera estratégica para mejorar la comprensión y la experiencia del usuario.

Para mantener la aplicación lo más actualizada posible contará con la integración de unas de APIs de datos, de forma que pueda acceder dinámicamente a datos, servicios o funcionalidades externas asegurando así una experiencia de usuario siempre actualizada y en sintonía con las últimas innovaciones y cambios en el entorno tecnológico.

En resumen, el objetivo de este proyecto es ayudar a los usuarios a mantenerse informados y actualizados sobre la seguridad vial, permitiendo escanear señales de tráfico, y mantenerse informados de las últimas noticias relacionadas. Todo ello gracias a una interfaz intuitiva, para garantizar una experiencia eficiente y sencilla.

1.2. REQUISITOS

Hemos identificado los siguientes requisitos indispensables que debe cumplir la aplicación a desarrollar:

- **Plataforma**: La aplicación debe ser compatible con varias plataformas móviles, tanto Android como iOS.
- **Integración de Cámara y Galería**: La aplicación debe integrar la funcionalidad de la cámara del dispositivo para el escaneo en tiempo real.
- **Interfaz de Usuario Intuitiva**: La interfaz de usuario debe ser fácil de usar y comprensible para los usuarios, con elementos visuales que mejoren la experiencia. Para ello seguiremos el principio 'WYSIWYG' (what you see is what you get).
- **Conexión a APIs Externas**: Para obtener información actualizada, la aplicación debe conectarse a APIs externas que proporcionen datos sobre señales de tráfico y noticias relevantes.
- **Compatibilidad con Múltiples Resoluciones**: La aplicación debe ser compatible con diferentes tamaños de pantalla y resoluciones de dispositivos móviles.

1.3. HERRAMIENTAS UTILIZADAS

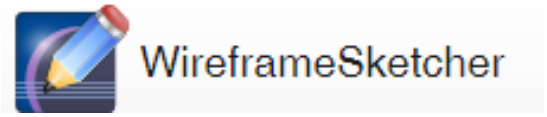
- **WireframeSketcher**
  Es una herramienta de diseño y prototipado que permite crear rápidamente bocetos de interfaces de usuario para aplicaciones y sitios web en las etapas iniciales del desarrollo. Facilita la visualización y comunicación de ideas antes de la implementación completa.
- **Flutter**
  Es un framework de código abierto desarrollado por Google que se utiliza para crear aplicaciones móviles nativas. Flutter utiliza como lenguaje de programación **Dart**, que es un lenguaje moderno y orientado a objetos desarrollado por Google. Además, una de sus principales ventajas es que permite crear aplicaciones con una única base de código que pueden ejecutarse en múltiples plataformas, incluyendo IOS, Android, web y escritorio.
- **APIs**
  Se han usado **dos APIs de datos**, además de una **base de datos en formato JSON** diseñada internamente para complementar con las funcionalidades de la aplicación. El uso de estas tres fuentes de datos combinadas proporciona una amplia cantidad de información, garantizando así la disponibilidad de datos precisos y relevantes para la aplicación.
  Las APIs utilizadas son las siguientes:
  - **NewsAPI (https://newsapi.ai/api/v1/article/getArticles)**
    Mediante esta API hemos obtenido las noticias relacionadas con el tráfico y la seguridad vial.
  - **Roboflow (https://detect.roboflow.com/)**
    Mediante esta API permitimos escanear la señal deseada, y obtener toda la información de ella. Esta API está basada en un modelo de visión por computador que implica identificar y localizar la presencia de una serie de objetos específicos de una imagen, en este caso, señales de tráfico. Para ello, utiliza algoritmos y modelos de aprendizaje automático.

Por otro lado, dado que no hemos identificado ninguna API de datos de señales de tráfico gratuita, ha sido necesario crear una base de datos personalizada para satisfacer este requisito. Por ello, se ha creado una base de datos, en formato JSON, en la que recogemos los principales datos de las señales de tráfico.

- **Postman**
  Postman es una herramienta de colaboración y desarrollo de API que permite a los usuarios enviar solicitudes HTTP Y HTTPS a una API para probar su funcionalidad. Así pues, se ha usado Postman para diseñar, probar, depurar y documentar las APIs de manera eficiente, facilitando el proceso de implementación de las APIs.

- **GitHub**
  Se ha usado GitHub para el control de versiones, el seguimiento de errores, la documentación de proyectos y el alojamiento de código fuente. Además de para el trabajo colaborativo de ambos participantes del proyecto.

  https://github.com/gvraul8/signalscan



2. DISEÑO

En la fase de diseño de nuestra aplicación, el principal objetivo es establecer la arquitectura de la aplicación, además de proporcionar una primera idea sobre la interfaz de usuario de manera intuitiva y accesible que satisfaga los requisitos y necesidades de los usuarios.

2.1. BOCETOS INICIALES

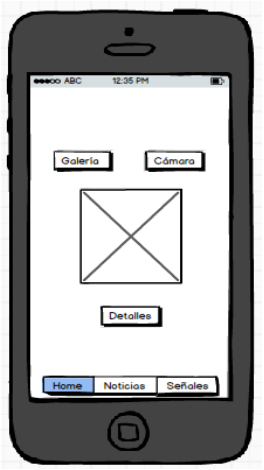
Los bocetos iniciales representan las primeras ideas y conceptos del diseño de la aplicación, sirviendo como base para el desarrollo de todo el proyecto. No obstante, **cabe recalcar que son bocetos, por lo que el diseño final podría sufrir algunas modificaciones.**
Así pues, una vez que hemos establecido los requisitos de la aplicación y las tecnologías a usar, pondremos énfasis en el diseño de las pantallas y la ubicación de los elementos de control de la aplicación. Para ello, hemos utilizado la herramienta **WireframeSketcher**.

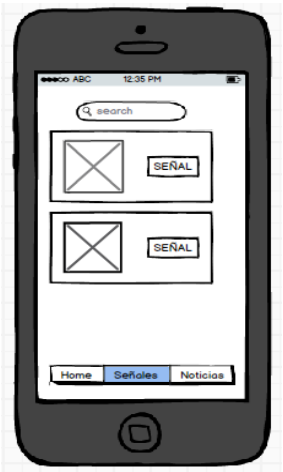
Como resultado de esta fase, hemos obtenido las siguientes pantallas:

- **Ventana de Inicio:**

  Esta ventana es la que se muestra nada más entrar a la aplicación. En ella se dispone de un menú de navegación para transitar entre las diversas interfaces. Además, en esta pantalla encontramos la función principal de esta aplicación, escanear fotos importadas desde la galería o la cámara.

  Se podrá volver a esta ventana pulsando la primera de las opciones seleccionables del menú de navegación.





- **Ventana de señales:**

Esta ventana se trata de la pantalla de señales. En ella podremos ver el listado de las señales de tráfico guardadas en nuestra base de datos.

Se podrá volver a esta ventana pulsando la segunda de las opciones seleccionables del menú de navegación.

- **Ventana de noticias:**

  Esta ventana se trata de la pantalla de noticias. En ella podremos deslizar hacia abajo para ver todas las noticias disponibles relacionadas con las señales de tráfico y la seguridad vial. Aparecen una foto de la noticia, junto a su autor o tema de esta y fecha.

  Se podrá volver a esta ventana pulsando la tercera de las opciones seleccionables del menú de navegación.

2.2. ESTUDIO DE LAS APIs UTILIZADAS

Como se ha mencionado, para el desarrollo de la aplicación necesitábamos obtener datos en tiempo real, para que estuvieran lo más actualizado posible. La solución a esto ha sido el uso de dos APIs: una para identificar y obtener información sobre los objetos de una imagen y otra para obtener las noticias relacionadas con las señales y seguridad vial.

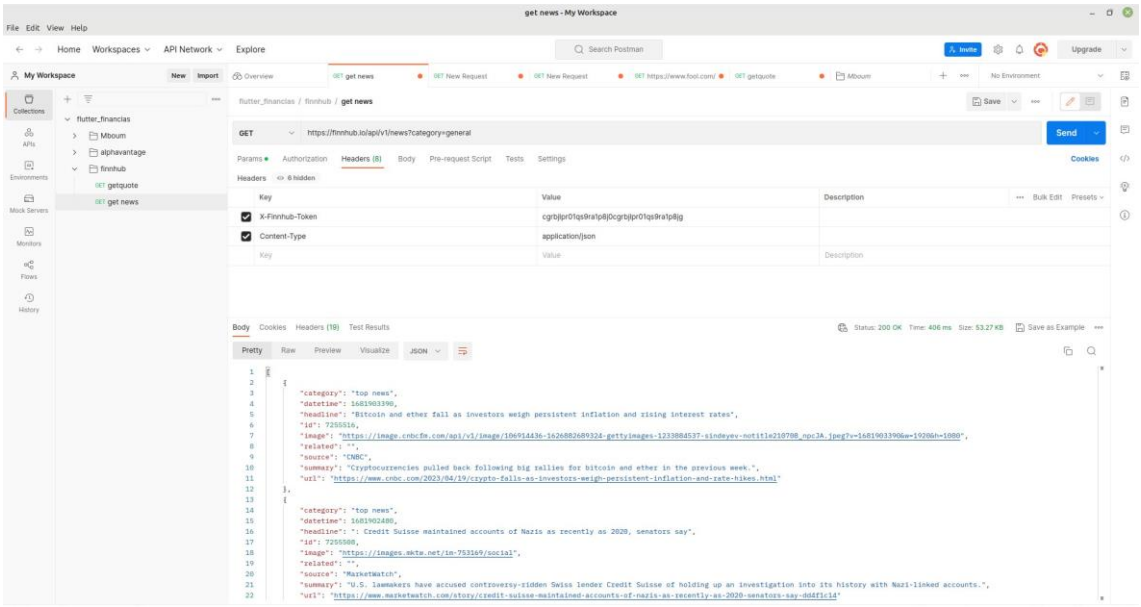
La **API de detección de objetos**, en este caso señales de tráfico, es de la página **(https://detect.roboflow.com/).** Esta usa un modelo de visión de computación y aprendizaje automático entrenado para identificar los objetos.

Para ello, ha sido necesaria una API KEY, que se pasará como parámetros, además, tendremos que especificar en las cabeceras (Headers) el Content-Type necesario, para nuestro caso, como nos indicaba la documentación de la API ha sido application/x-www-form-urlencoded, y por último será necesaria pasarle en el Body la imagen codificada en Base64.

Así pues, obtendremos los siguientes datos como respuesta:

- "time": el tiempo que ha tardado en procesar la imagen
- "image": un objeto de tipo image con las dimensiones originales de la imagen proporcionada.
- "predictions": una lista de objetos predicciones, que será los resultados obtenidos al identificar la imagen, que contienen los siguientes parámetros:
    - "x" e "y": coordenadas (x,y) del objeto identificado.
    - "width" y "height": tamaño del objeto identificado.
    - "confidence": valor (entre 0 y 1) de la coincidencia con el objeto identificado.
    - "class": clase a la que pertenece el objeto identificado
    - "class_id": id de la clase

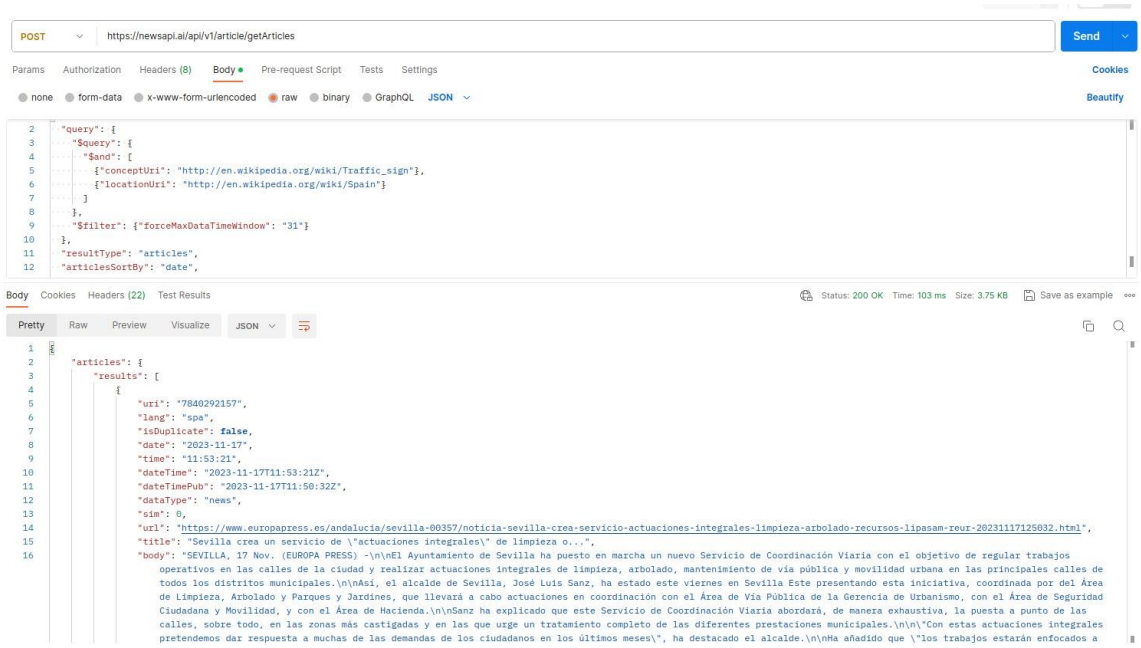
Este es el resultado obtenido en Postman:



Por otro lado, la API de noticias de señales de tráfico y seguridad vial es de la página **https://newsapi.ai/api/v1/article/getArticles**. Esta es más sencilla que la anterior, ya que sólo ha sido necesaria el uso de una API KEY, que por su parte y desventaja para nosotros, es de coste y con la versión gratuita estamos limitados a sólo 2000 llamadas. Además, en el Body ha sido necesario especificar el tema sobre el que queríamos noticias y el país.

Como resultado obtenemos una lista de objetos de tipo “articles”, en los que nos proporciona información como fecha, url, titulo, autor, etc.

Como ejemplo, esta llamada desde Postman:



3. IMPLEMENTACIÓN

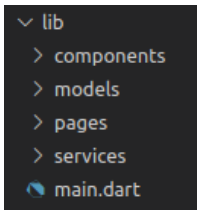
La implementación de un proyecto es una de las fases más importantes dentro del ciclo de vida del software. En este sentido, el objetivo de esta memoria es describir la implementación de un proyecto en el cual se ha utilizado Flutter como framework para el desarrollo de la aplicación móvil.

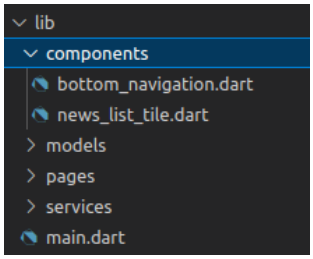
Como se ha mencionado, Flutter es un framework de código abierto creado por Google, que permite el desarrollo de aplicaciones móviles para Android e iOS de forma rápida y eficiente, utilizando un lenguaje de programación único: Dart.

3.1. ORGANIZACIÓN DEL PROYECTO

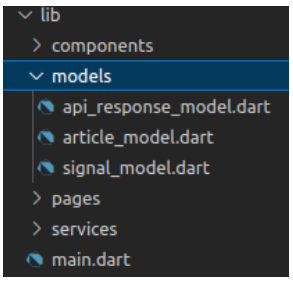
Se ha considerado que la estructuración del proyecto carpetas ayuda a mantener una separación clara y organizada de la lógica del software, lo que facilita el mantenimiento y la escalabilidad del proyecto en el futuro.

El proyecto se ha estructurado en cuatro carpetas principales: components, models, pages y services. Cada una de estas carpetas tiene una función específica en la organización del código y la lógica del software.

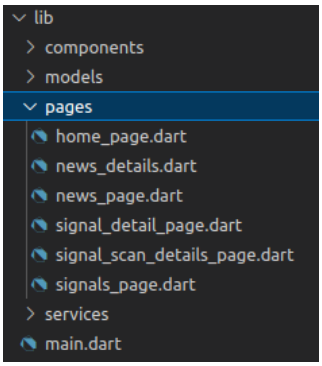




La carpeta "components" contiene todos los widgets personalizados y reutilizables que se utilizan en la interfaz de usuario de la aplicación. Estos componentes incluyen los botones del menú de navegación (bottom_navigation.dart) y el estilo de las lista de noticias (news_list_tile.dart).

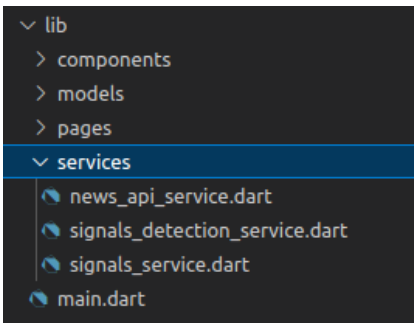


La carpeta "models" contiene las definiciones de los modelos de datos que se utilizan en la aplicación. Estos modelos pueden ser objetos que representan estructuras de datos que se utilizan para transmitir información entre diferentes partes de la aplicación. En este caso, usamos tres modelos, uno para el modo de representación de las listas de noticias (article_model.dart), otro para la lista de cotizaciones (signal_model.dart) y otro para controlar la respuesta de la API de detección de objetos (api_response_model.dart).



La carpeta "pages" contiene todas las páginas de la aplicación. Cada página se implementa como un widget de Flutter que define la estructura y el diseño de la interfaz de usuario de esa página. Las páginas pueden incluir widgets de componentes y servicios para proporcionar funcionalidades específicas. Las páginas en las que hemos dividido nuestra aplicación son las siguientes: una página para el inicio (home_page.dart), para el apartado de noticias tenemos dos páginas, una para el listado de noticias (news_page.dart) y otra para ver los detalles de una noticia concreta (news_details_page.dart). Al igual que para las noticias, encontramos dos páginas para las señales (signal_details_page.dart) y (signals_page.dart) para ver los detalles de las señales y el listado de señales respectivamente. Por último, tenemos una página (signal_scan_details_page.dart) para ver los detalles del escaneo de la imagen.

La carpeta "services" contiene las definiciones de los servicios que se utilizan en la aplicación. Estos servicios son las conexiones a las APIs usadas. Así pues, observamos una página para cada API (news_api_service.dart y signals_detections_service.dart), además de otra página para la base de datos en JSON que hemos utilizado (signals_service.dart).



La base de datos en JSON la encontramos en la carpeta assets/signals, en la que se encuentra los datos en json junto a las imágenes necesarias.

3.2. ASPECTOS DESTACADOS DE LA IMPLEMENTACIÓN

En este apartado, se detallarán los aspectos más importantes de la implementación del proyecto en Flutter. Se mencionarán algunas de las clases y métodos más relevantes utilizados en el desarrollo de la aplicación móvil, con el objetivo de destacar las características y funcionalidades más destacadas del software. No obstante, si existe la curiosidad de algún código no mencionado en esta memoria cabe destacar que el proyecto está disponible en Github, como se mencionó anteriormente.

Así pues, hemos considerado que uno de los apartados más importantes son las llamadas a las APIs. Puesto que, como se ha mencionado, disponemos de varias APIs, no todas las llamadas son iguales, por lo que se comentará de manera superficial las partes más importantes del código.

Empecemos observando el código de la API de noticias. En primer lugar, definimos una serie de constantes, las cuales como se han mencionado en el apartado de APIs utilizadas son la propia url de la API, “apiUrl, la cabecera, “headers”, que en este caso solo hace falta especificar el ‘Content-Type’ y el body, “body”, en el es necesario indicar una serie de parámetros según la documentación de la API, en este caso destacamos API KEY, y el concepto o localización de la búsqueda que estamos haciendo.

```
class NewsAPIService {
  Future<List<Article>> getNews() async {
    const apiUrl = "https://newsapi.ai/api/v1/article/getArticles";
    final headers = {'Content-Type': 'application/json'};
    final body = {
      'query': {
        '\$query': {
          '\$and': [
            {"conceptUri": "http://en.wikipedia.org/wiki/Traffic_sign"}
            {"locationUri": "http://en.wikipedia.org/wiki/Spain"}
          ]
        },
        '\$filter': {"forceMaxDataTimeWindow": "31"}
      },
      "resultType": "articles",
      "articlesSortBy": "date",
      "apiKey": "9758aa6b-d20f-4ef3-b173-5c8dd728f176"
    };
```

Para controlar estos datos recibidos por la API (recomendamos ver el apartado de APIs utilizadas en el que se muestra un ejemplo de Postman de los datos recibimos) es necesario crear un objeto, nosotros le hemos llamado Article, y contiene los siguientes parámetros:

```
class Article {
  String date;
  String url;
  String title;
  String body;
  Source source;
  String image;
  int relevance;

  Article({
    required this.date,
    required this.url,
    required this.title,
    required this.body,
    required this.source,
    required this.image,
    required this.relevance,
  });

  factory Article.fromJson(Map<String, dynamic> json) {
    return Article(
      date: json['date'],      You, 7 days ago • first comm
      url: json['url'],
      title: json['title'],
      body: json['body'],
      source: Source.fromJson(json['source']),
      image: json['image'],
      relevance: json['relevance'],
    ); // Article
  }
}
```

Para la representación de estos datos, se usa el archivo news_page.dart, que crea la página principal y llama al archivo news_list_tile.dart, mediante el método NewsListTile() (Codigo 2), que básicamente se encarga de dar el formato con el que aparecen las noticas. En este método, se controla también que si pulsamos sobre una noticia, nos aparezca la venta de detalles de la noticia, es decir aparece la news_details_page.dart. En este último archivo, se encuentra el botón de ver más y el código correspondiente para la redirección al URL de la noticia (Codigo 3).

***Por simplicidad, sólo se van a poner algunos ejemplos de código, para visualizar todo el código completo mirar el repositorio de Github habilitado (**https://github.com/gvraul8/signalscan)**.**

```dart
class NewsPage extends StatelessWidget {
  const NewsPage({super.key});

  @override
  Widget build(BuildContext context) {
    NewsAPIService client = NewsAPIService();
    return Scaffold(
        appBar: AppBar(
          backgroundColor: const Color.fromARGB(255, 165, 36, 36),
          title: const Text('Noticias'),
        ), // AppBar
        body: FutureBuilder(
          future: client.getNews(),
          builder:
              (BuildContext context, AsyncSnapshot<List<Article>> snapshot) {
            if (snapshot.hasData) {
              List<Article>? articles = snapshot.data;
              return ListView.builder(
                  itemCount: articles?.length,
                  itemBuilder: (context, index) =>
                      NewsListTile(articles![index], context)); // ListView.builder
            }

            return const Center(
              child: CircularProgressIndicator(),
            ); // Center
          },
        )); // FutureBuilder // Scaffold
  }
}
```

```dart
Widget NewsListTile(Article article, BuildContext context) {
  return InkWell(
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => NewsDetailsPage(article: article)),
      );
    },
    child: Container(
      margin: const EdgeInsets.all(12.0),
      padding: const EdgeInsets.all(8.0),
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(10.0),
        boxShadow: const [
          BoxShadow(
            color: Colors.black12,
            blurRadius: 5.0,
            spreadRadius: 1.0,
            offset: Offset(1.0, 1.0),
          ) // BoxShadow
        ],
      ), // BoxDecoration
```

La llamada a la API de detección de objetos es similar a la anterior, con la diferencia de que la imagen que le proporcionamos debe estar codificada en Base64. Por lo demás, es necesario que le pasemos la API KEY, la cabecera con el 'Content-Type', y como body la imagen codificada.

```dart
const String apiUrl = "https://detect.roboflow.com/ts-v5/1";
const String apiKey = "ftKgIkOFKpkA8w6AP6Tt";

You, 5 days ago | 1 author (You)
class ApiService {
  static Future<ApiResponse> sendImageForDetection(File imageFile) async {
    // Lee los bytes del archivo
    List<int> imageBytes = await imageFile.readAsBytes();

    // Convierte los bytes a base64
    String base64Image = base64Encode(imageBytes);

    final response = await http.post(
      Uri.parse("$apiUrl?api_key=$apiKey"),
      body: base64Image,
      headers: {"Content-Type": "application/x-www-form-urlencoded"},
    );

    if (response.statusCode == 200) {
      // Si la solicitud es exitosa, analiza la respuesta JSON
      Map<String, dynamic> responseData = json.decode(response.body);
      return ApiResponse.fromJson(responseData);
    } else {
      // Si la solicitud no fue exitosa, lanza una excepción.
      throw Exception('Error al cargar datos desde la API');
    }
  }
}
```

Sin embargo, como se puede observar en el apartado anterior en el que se proporciona un ejemplo con Postman de la respuesta obtenida, podemos observar como está se complica respecto a la respuesta de la API de noticias. Es por ello, por lo que necesitamos definir varias clases, para poder controlar de forma correcta esta respuesta. Dos de las clases más importantes son ImageInfo, que nos permitirá posteriormente mostrar la imagen en la cual estamos buscando señales, y la clase Prediction, la cual recogerá los datos de los objetos identificados en la imagen proporcionada.

```dart
class Prediction {
  final double x;
  final double y;
  final double width;
  final double height;
  final double confidence;
  final String classLabel;
  final int classId;

  Prediction({
    required this.x,
    required this.y,
    required this.width,
    required this.height,
    required this.confidence,
    required this.classLabel,
    required this.classId,
  });

  factory Prediction.fromJson(Map<String, dynamic> json) {
    return Prediction(
      x: json['x'].toDouble(),
      y: json['y'].toDouble(),
      width: json['width'].toDouble(),
      height: json['height'].toDouble(),
      confidence: json['confidence'].toDouble(),
      classLabel: json['class'],
      classId: json['class_id'],
    );
  }
```

```dart
class ImageInfo {
  final int width;
  final int height;

  ImageInfo({required this.width, required this.height});

  factory ImageInfo.fromJson(Map<String, dynamic> json) {
    return ImageInfo(
      width: json['width'],
      height: json['height'],
    );
  }
}
```

Sin embargo, la dificultad con esta API no termina aquí, ya que como hemos visto la API nos ofrece las coordenadas x e y del objeto identificado, junto a su tamaño, sin embargo para una buena representación visual es necesario remarcar esto sobre la foto mediante un cuadrado en estas coordenadas y con este tamaño indicado, todo ello, teniendo en cuenta que pueden existir varias identificaciones en una misma foto, es decir, que existe la posibilidad y la necesidad de imprimir varios cuadrados en una misma foto. A esto, se le suma la dificultad de que no todas las imágenes son del mismo tamaño, es más, no todos los dispositivos tienen el mismo de tamaño de pantalla, por lo que será necesario controlar esto.

Para ello, creamos una lista para almacenar todas las predicciones detectadas, y posteriormente seguimos estos pasos:

- En primer lugar, leemos el tamaño de la imagen recibida.
- En segundo lugar, comprobamos la resolución de pantalla que estamos usando.
- En tercer lugar, con los dos datos anteriores, calculamos una escala para redimensionar la imagen.
- En cuarto lugar, usando la escala calculada anteriormente, recalculamos tanto las coordenadas x e y como el tamaño el cuadrado a dibujar en el objeto detectado.

```
List<PredictionValues> predictionsValuesList = [];
for (var i = 0; i < apiResponse.predictions.length; i++) {
  var prediction = apiResponse.predictions[i];
  double x = prediction.x;
  double y = prediction.y;
  double confidence = prediction.confidence;
  String className = prediction.classLabel;

  // Dimensiones reales de la imagen
  double originalImageWidth = apiResponse.image.width.toDouble();
  double originalImageHeight = apiResponse.image.height.toDouble();

  // Ancho deseado de la imagen (ajustado al ancho de la pantalla)
  double desiredImageWidth = MediaQuery.of(context).size.width * 0.85;

  // Calcular la altura para mantener la relación de aspecto
  double desiredImageHeight =
      originalImageHeight * (desiredImageWidth / originalImageWidth);

  // Factor de escala para ajustar el cuadrado al tamaño de la imagen
  double scale = desiredImageWidth / originalImageWidth;

  // Ajusta las coordenadas x e y proporcionalmente
  x = scale * (x - (prediction.width / 2));
  y = scale * (y - (prediction.height / 2)) +
      (MediaQuery.of(context).size.width * 0.85 - desiredImageHeight) / 2;

  // Ajusta el tamaño del cuadrado proporcionalmente
  double width = prediction.width * scale;
  double height = prediction.height * scale;
```

Tras esto, añadimos la predicción a la lista de predicciones para representarla posteriormente.

```
predictionsValuesList.add(PredictionValues(
  x: x,
  y: y,
  confidence: confidence,
  className: className,
  width: width,
  height: height,
  number: i + 1, // Número de la predicción
));
```

Una vez añadido todas las predicciones a la lista, el siguiente paso será mostrar por pantalla los resultados, es decir la signal_scan_details_page.dart.

```
for (var values in predictionsValuesList)
  Positioned(
    left: values.x,
    top: values.y,
    child: Column(
      children: [
        Container(
          width: values.width,
          height: values.height,
          decoration: BoxDecoration(
            border: Border.all(
              color: getColorForConfidence(values.confidence),
              width: 2.0,
            ), // Border.all
          ), // BoxDecoration
        ), // Container
        Container(
          margin: const EdgeInsets.only(top: 2),
          decoration: BoxDecoration(
            color: Colors.black,
            borderRadius: BorderRadius.circular(4),
          ), // BoxDecoration
          padding: const EdgeInsets.symmetric(
              horizontal: 4, vertical: 2), // EdgeInsets.symmetric
          child: Text(
            values.number.toString(),
            style: const TextStyle(
              color: Colors.white,
              fontSize: 12,
            ), // TextStyle
          ), // Text
```

Siendo el getColorForConfidence un método el cual dibujará el cuadrado de un color u otro dependiendo de la similitud encontrada.

```
Color getColorForConfidence(double confidence) {
  if (confidence > 0.9) {
    return Colors.green;
  } else if (confidence >= 0.7 && confidence <= 0.9) {
    return Colors.yellow;
  } else {
    return Colors.red;
  }
}
```

Por otro lado, la lectura de los datos de la base de datos personalizada que hemos creado para esta aplicación es mucho más sencilla.

```
class SignalsService {
  Future<List<Signal>> getSignals() async {
    try {
      String signalsData = await rootBundle.loadString('assets/signals/signals.json');
      List<dynamic> jsonList = json.decode(signalsData);

      List<Signal> signals = jsonList
          .map((json) => Signal(
                name: json['name'],
                image: json['image'],
                meaning: json['meaning'],
              ))
          .toList();

      return signals;
    } catch (e) {
      print('Error cargando señales: $e');
      return [];
    }
  }
}
```

Por último, para que aparezca el menú de navegación en el inferior de la pantalla se llama al archivo bottom_navigation_bar.dart mediante el método BaseApp desde el archivo main.dart.

```dart
final List<BottomNavigationBarItem> _bottomNavigationBarItems = const [
  BottomNavigationBarItem(
    icon: Icon(Icons.home),
    label: 'Home',
  ), // BottomNavigationBarItem
  BottomNavigationBarItem(
    icon: Icon(Icons.newspaper),
    label: 'Noticias',
  ), // BottomNavigationBarItem
  BottomNavigationBarItem(
    icon: Icon(Icons.traffic),
    label: 'Señales',
  ), // BottomNavigationBarItem
];

static const List<Widget> _widgetOptions = <Widget>[
  HomePage(),
  NewsPage(),
  SignalsPage()
]; // <Widget>[]
```

```dart
Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

You, last week | 1 author (You)
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'SignalScan',
      debugShowCheckedModeBanner: false,
      home: BaseApp(),
    ); // MaterialApp
  }
}
```

3.3. RESULTADO FINAL

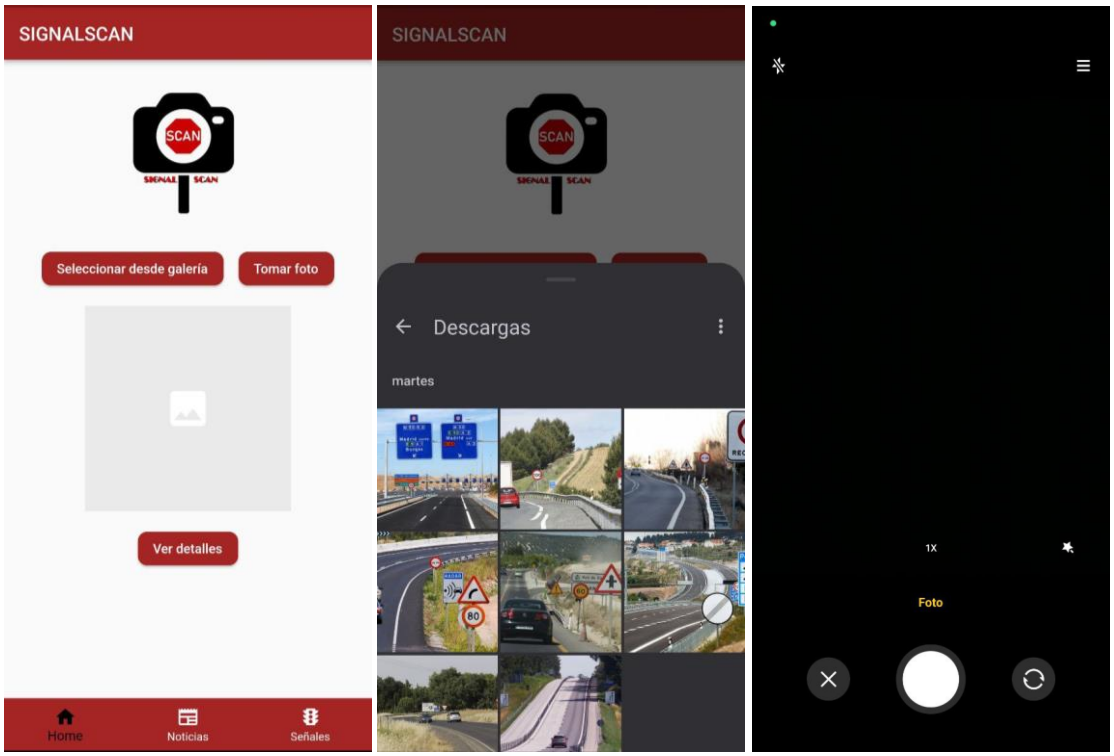
- **Ventana Inicio**

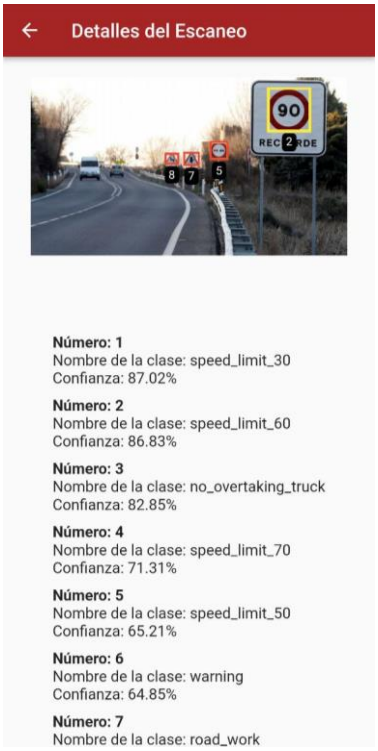
En la ventana de inicio encontramos en primer lugar el icono de la aplicación. Debajo de este encontramos dos botones “Seleccionar desde galería” y “Tomar foto”. Si pulsamos en el primero de ello se nos abrirá la galería del teléfono para que seleccionemos una imagen. En el caso de pulsar sobre el segundo de los botones se nos abrirá la propia aplicación de la cámara del dispositivo.

En el centro de la pantalla aparecerá la imagen seleccionada con los botones anteriores. Por defecto, al no haber ninguna imagen seleccionada aparece el icono de una imagen.

Debajo de esto, encontramos el botón ver detalles, que al pulsar sobre él nos abrirá una nueva página, donde aparecerá la imagen que hemos seleccionado con los objetos detectados marcados con cuadrados y numerados, permitiendo así identificar de manera fácil que descripción se corresponde con su objeto identificado. Así podemos hacer scroll para ver todas las identificaciones.

Se puede volver en cualquier momento a esta ventana pulsando sobre el icono de “Home” del menú de navegación situado en la parte inferior de la pantalla.





- **Ventana de Noticias**

Para acceder a esta ventana es necesario pulsar sobre el icono de “Noticias” que nos encontramos en el menú de navegación situado en el inferior de nuestra pantalla.

Una vez accedamos a esta pantalla, nos encontraremos un listado de noticias que irá ampliando de forma que vayamos deslizando de forma vertical. En esta pantalla observamos las noticias por su imagen, titular y fuente de la noticia.

Podemos acceder a más información de una noticia pulsando sobre ella.

En esta pantalla encontramos un botón de “Leer más”, que si pulsamos sobre él nos ampliará la información que recibimos de la noticia, que podremos leer haciendo scroll. También disponemos de un botón “Ver en detalle” que si pulsamos sobre él nos redirigirá a la página de la noticia para poder leerla de forma completa.

Noticias

europa press

Sevilla crea un servicio de "actuaciones integrales" de limpieza o...

Home Noticias Señales

Sevilla crea un servicio de ...

europa press

Sevilla crea un servicio de "actuaciones integrales" de limpieza o...

SEVILLA, 17 Nov. (EUROPA PRESS) -

El Ayuntamiento de Sevilla ha puesto en marcha un nuevo Servicio de Coordinación Viaria con el objetivo de regular trabajos operativos en las calles de la ciudad y r...

Leer más Ver en detalle

Sevilla crea un servicio de ...

todos los distritos municipales.

Así, el alcalde de Sevilla, José Luis Sanz, ha estado este viernes en Sevilla Este presentando esta iniciativa, coordinada por del Área de Limpieza, Arbolado y Parques y Jardines, que llevará a cabo actuaciones en coordinación con el Área de Vía Pública de la Gerencia de Urbanismo, con el Área de Seguridad Ciudadana y Movilidad, y con el Área de Hacienda.

Sanz ha explicado que este Servicio de Coordinación Viaria abordará, de manera exhaustiva, la puesta a punto de las calles, sobre todo, en las zonas más castigadas y en las que urge un tratamiento completo de las diferentes prestaciones municipales.

"Con estas actuaciones integrales pretendemos dar respuesta a muchas de las demandas de los ciudadanos en los últimos meses", ha destacado el alcalde.

Ha añadido que "los trabajos estarán enfocados a aumentar el nivel de limpieza en las calles y, en concreto, en determinados puntos negros con baldeos de agua a presión y productos biodegradables; a asfaltado y señales de tráfico; a tareas de desbroce; a repaso de señalética y bolardos; a eliminación de pinturas, grafitis y cartelería en paredes, farolas y demás mobiliario urbano; y a limpieza de papeleras y contenedores".

En este sentido, el mayor número de efectivos procederán de la empresa municipal de limpieza, Lipasam. El dispositivo total contará también con operarios de los equipos de trabajo de REUR (Servicio de Respuesta Urbana), y de los servicios de Parques y Jardines y Protección Ambiental, Movilidad y Vía Pública.

El alcalde de Sevilla ha concluido que "la limpieza es una de mis prioridades, me comprometí a cuidar Sevilla y a incrementar la limpieza de la ciudad y eso es lo que estamos haciendo de forma coordinada y planificada, introduciendo mejoras y actuaciones constantes, con el único fin de que Sevilla y las calles de todos los barrios estén limpias".

Leer menos Ver en detalle

europapress.es

europa press

SEVILLA

Sevilla crea un servicio de "actuaciones integrales" de limpieza o arbolado con recursos de Lipasam o REUR

Archivo - Operarios limpian en las inmediaciones del colegio del barrio de San Jerónimo

- FRANCISCO J. OLMO - EUROPA PRESS - ARCHIVO

Europa Press Andalucía Publicado: 17/11/2023 12:50

Newsletter @epandalucia

- **Ventana de señales**

Para acceder a esta ventana es necesario pulsar sobre el icono de “Señales” que nos encontramos en el menú de navegación situado en el inferior de nuestra pantalla.

En ella nos encontramos un listado de señales con su imagen y nombre. Además, si pulsamos sobre el icono de la parte superior derecha podemos modificar como ver el listado, de 1 en 1, o de 2 en 2.

Para observar la lista completa se puede, al igual que en el apartado de noticias, deslizar de forma vertical.

Además, si pulsamos sobre alguna de las señales podemos ver más detalles y obtener una pequeña descripción de que quiere decir la señal.