ChatGPT

Parkit: Aplicación de Estacionamiento Compartido en Flutter

Parkit es una app móvil colaborativa para conductores en Argentina que ayuda a encontrar lugares libres para estacionar en la vía pública mediante Google Maps y geolocalización. La interfaz se diseñará con **estilo moderno y flat**, usando colores celeste/azul inspirados en la señalización de estacionamiento argentina, fuentes limpias (por ejemplo Roboto) y elementos de UI con bordes redondeados y sombras suaves. La **pantalla de bienvenida** mostrará el logo "parkit" centrado sobre fondo celeste/azul con una animación de entrada suave. La **pantalla principal** mostrará un `GoogleMap` ocupando todo el espacio: los lugares libres aparecerán como PINS verdes y clickeables. Habrá un botón flotante centrado abajo para abrir el menú (perfil, configuración, cerrar sesión, Términos). Se usarán animaciones sutiles (transiciones, sombras, `AnimatedContainer`) para una experiencia fluida. Se seguirá el sistema Material Design de Flutter, seleccionando un color primario celeste y secundario blanco/azul, usando `ThemeData` personalizado para mantener consistencia visual.

Ejemplo ilustrativo de interfaz de app de estacionamiento con mapas (UI inspiracional).

Estructura del Proyecto y Configuración

- **Crear proyecto Flutter:** Iniciar con `flutter create parkit_app`.
- **Dependencias:** En `pubspec.yaml` agregar paquetes clave:

```
dependencies:
  flutter:
    sdk: flutter
  provider: ^6.0.5            # o Riverpod para manejo de estado
  supabase_flutter: ^1.4.0    # Cliente de Supabase (Auth, DB)
  google_maps_flutter: ^2.2.0 # Widget de mapas
  sensors_plus: ^6.1.1        # Acceso a acelerómetro/giroscopio [1]
  geolocator: ^14.0.1         # Ubicación GPS [2]
  flutter_local_notifications: ^19.2.1  # Notificaciones locales [3]
```

- **Claves y permisos:**
- **Supabase:** Crear un proyecto en Supabase (con la API `URL` y clave pública `anon key`). Copiar estos valores a constantes Dart o a un archivo de configuración. Inicializar en `main.dart`:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Supabase.initialize(
    url: 'https://<TU-PROYECTO>.supabase.co',
    anonKey: '<TU-ANON-KEY>',
```

```
  ); // Inicializa Supabase [4]
  runApp(const MyApp());
}
```

- **Google Maps:** Conseguir API Key para Android/iOS. En AndroidManifest (y Info.plist en iOS) agregar `<meta-data android:name="com.google.android.geo.API_KEY" android:value="TU_API_KEY"/>`.
- **Permisos:** Solicitar permisos de ubicación y sensores (agregar descripciones en Info.plist para iOS, en Android manifest permisos GPS, etc).
- **Tema y fuentes:** Configurar `ThemeData` global con `primarySwatch: Colors.lightBlue` y `fontFamily: 'Roboto'`.

Autenticación y Usuarios

Se implementa **registro/ingreso por email y contraseña** usando Supabase Auth. En la pantalla de *Login* y *Signup* se usan formularios sencillos. Al registrarse, el código Dart llama a:

```
final AuthResponse res = await Supabase.instance.client.auth.signUp(
  email: email, password: password
);
```

Al hacer login:

```
final AuthResponse res = await Supabase.instance.client.auth.signInWithPassword(
  email: email, password: password
);
```

Este flujo está soportado por Supabase de forma segura [5] [6]. Tras autenticar, se guarda la sesión y se redirige al usuario a la pantalla principal del mapa. También habrá una pantalla de perfil donde el usuario puede ver/editar sus datos (nombre, email) y datos del auto. Estos datos (en especial el auto) se guardan en la base de datos Supabase.

Perfil del Auto

En el **perfil del auto**, el usuario ingresa `marca`, `modelo` y `año`. Para mayor comodidad, se sugiere conectar con una *API (falsa para MVP)* que, dado marca/modelo/año, retorne las dimensiones (`largo`, `ancho`). Por ejemplo, al seleccionar modelo, se podría llamar a un servicio REST (o base de datos local) que complete automáticamente `largo_cm` y `ancho_cm`. Estos datos se almacenan en la tabla `autos` de Supabase. Por ejemplo:

```
await supabase.from('autos').insert({
  'marca': marcaSeleccionada,
  'modelo': modeloSeleccionado,
```

```
  'anio': anioSeleccionado,
  'largo_cm': largoRecibido,
  'ancho_cm': anchoRecibido
});
```

Luego, se vincula este `auto_id` al perfil del usuario (`users.auto_id`). Cada vez que el conductor comparta un lugar, se usarán las dimensiones de su auto desde esta tabla para calcular recomendaciones.

Mapa con Lugares Libres

La pantalla principal muestra un mapa con el widget `GoogleMap` de Flutter. Al iniciar, se ubica la cámara en la posición del usuario (obtenida con geolocator) o en la ciudad de Buenos Aires. Los **lugares libres** se obtienen de la tabla `estacionamientos` filtrando `status = 'libre'`. Por cada lugar libre, se crea un `Marker` verde con un tamaño proporcional. Ejemplo básico:

```
GoogleMap(
  initialCameraPosition: CameraPosition(target: latLngCentral, zoom: 15),
  markers: Set<Marker>.from(_marcadoresLibres), // marcadores obtenidos de DB
  onMapCreated: (controller) { _controller = controller; _cargarMarcadores(); },
);
```

La función `_cargarMarcadores()` hace:

```
final data = await supabase.from('estacionamientos').select().eq('status',
'libre');
_setMarkers(data);
```

Cada `Marker` tiene un `onTap` que muestra un **info window personalizado** con detalle de medidas y autos recomendados. En Flutter es posible personalizar totalmente la ventana emergente usando widgets propios [7]. Por ejemplo, al pulsar un pin se abre un `showModalBottomSheet` o un widget en pantalla que muestra: - **Dimensiones** del lugar (largo x ancho). - **Lista de autos sugeridos**: aquellos cuyo `largo_cm` y `ancho_cm` (desde `autos` DB) son menores al lugar + un margen de seguridad. Esto permite filtrar autos que caben.

Así, el mapa interactivo muestra en tiempo real las plazas disponibles. Para una mejor UX, las marcas pueden crearse con widgets Flutter (por ejemplo usando `toBitmapDescriptor()` de un widget personalizado) para iconos con logos o texto [8]. Por ejemplo, un marcador de lugar libre puede ser un cuadrado verde con un icono de coche dibujado.

Compartir Lugar de Estacionamiento

Cuando el usuario estacione, pulsa un botón **“Compartir mi lugar”**. Esto hace lo siguiente:

1. **Obtener ubicación actual:** Usar `geolocator` para capturar latitud/longitud del dispositivo [2].

2. **Guardar como "ocupado":** Insertar en la tabla `estacionamientos` un nuevo registro con `user_id`, coordenadas, las dimensiones del auto (`largo_cm`, `ancho_cm`), `timestamp` y `status = 'ocupado'`. Ejemplo:

```
Position pos = await Geolocator.getCurrentPosition();
await supabase.from('estacionamientos').insert({
  'user_id': currentUser.id,
  'lat': pos.latitude,
  'lng': pos.longitude,
  'largo_cm': autoUsuario.largo,
  'ancho_cm': autoUsuario.ancho,
  'timestamp': DateTime.now().toIso8601String(),
  'status': 'ocupado'
});
```

3. **Detección de partida:** Se usa el *sensor de movimiento (giroscopio/acelerómetro)* para detectar cuándo el coche empieza a moverse. Con el paquete `sensors_plus` es posible suscribirse a eventos de aceleración [1]:

```
accelerometerEvents.listen((AccelerometerEvent event) {
  if (event.x.abs() + event.y.abs() + event.z.abs() > UMBRAL_MOVIMIENTO) {
    // Detectado movimiento
    _enviarNotificacionAvance();
  }
});
```

4. **Notificación al moverse:** Al detectar movimiento, se envía una notificación local (usando `flutter_local_notifications`) preguntando si el conductor dejó el lugar. Ejemplo:

```
final flutterLocalNotificationsPlugin = FlutterLocalNotificationsPlugin();
// Configurar notificación (canal, icono, etc) y mostrarla
var androidDetails = AndroidNotificationDetails('canal1', 'Parkit
Notifs', ...);
var generalNotificationDetails = NotificationDetails(android:
androidDetails);
flutterLocalNotificationsPlugin.show(
  0, '¿Dejaste el estacionamiento?', 'Pulsa para confirmar lugar libre.',
generalNotificationDetails
);
```

   De esta forma, cuando el usuario confirma haber dejado el lugar, se ejecuta un callback.
5. **Confirmación y actualización:** Al confirmar en la notificación, la app actualiza el registro en Supabase: cambia `status` a `"libre"`, y actualiza `timestamp`. De esta forma se crea una **"mancha libre"** con las dimensiones exactas del auto del usuario. Por ejemplo:

```
await supabase.from('estacionamientos')
  .update({'status': 'libre', 'timestamp':
DateTime.now().toIso8601String()})
  .eq('user_id', currentUser.id)
  .eq('status', 'ocupado');
```

(También se podría crear un nuevo registro, pero aquí se simplifica actualizando el mismo).

6. **Visualización en el mapa:** Finalmente, ese lugar pasa a aparecer como PIN verde libre en el mapa para que otros usuarios lo vean. Si otro usuario lo ocupa antes, el registro se marcará como "ocupado" de nuevo.

Este flujo permite un sistema colaborativo de plazas: los conductores "reportan" en tiempo real cuando se van, liberando lugares para la comunidad. Para asegurar consistencia, se pueden usar funciones (ver siguiente sección) que borren lugares no confirmados o antiguos.

Publicidad y Monetización

La app incluye publicidad contextual en forma de **globos/pins de marcas** en el mapa. Estos pins especiales no tapan los lugares de estacionamiento. Por ejemplo, se crea una tabla `publicidades` con campos `(id, marca, lat, lng, texto, url, activo)`. Al cargar el mapa, además de los pins verdes, se agregan `Marker`s personalizados (por ejemplo un icono de globo o logotipo de marca) en las coordenadas publicitadas. Se puede usar un `BitmapDescriptor` desde imagen de asset para cada marca.

Cuando el conductor se acerca a la zona (p.ej. dentro de 100m), el pin se expande suavemente: esto puede lograrse monitoreando la ubicación del usuario y, al estar cerca, mostrar un widget sobrepuesto o un InfoWindow ampliado con la marca. Al tocar el globo o banner, la app abre la URL en un navegador (landing page de la marca). Este flujo es similar a crear **Info Windows personalizados** en Google Maps, lo cual es sencillo en Flutter usando widgets propios [7]. En resumen: - Marcar anuncios activos en el mapa con iconos distintivos (p.ej. globos o logos de marcas).
- Al acercarse (`if (distanceToAd < radio)`) o al pulsar, mostrar detalles (texto e imagen de la marca).
- Mantener los marcadores publicitarios ligeramente translúcidos o con menor prioridad de z-index para no tapar completamente los pins verdes.
- Usar animaciones (`AnimatedContainer`, `Tween`) para expandir suavemente la vista del anuncio al ser relevante.

Base de Datos (Supabase)

Se usa Supabase (PostgreSQL) como backend. Se proponen estas tablas (esquema mínimo):

- **users**: `id (uuid PK), email (text, único), password_hash?, nombre (text), auto_id (uuid FK a autos.id)`.
- **autos**: `id (uuid PK), marca (text), modelo (text), anio (int), largo_cm (int), ancho_cm (int)`.

- **estacionamientos**: `id (uuid PK), user_id (uuid FK a users), lat (float), lng (float), largo_cm (int), ancho_cm (int), timestamp (timestamptz), status (text CHECK libre/ocupado)`.
- **publicidades**: `id (uuid PK), marca (text), lat (float), lng (float), texto (text), url (text), activo (boolean)`.

Ejemplo de creación SQL:

```sql
CREATE EXTENSION IF NOT EXISTS "pgcrypto"; -- para gen_random_uuid()

CREATE TABLE autos (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  marca TEXT NOT NULL,
  modelo TEXT NOT NULL,
  anio INTEGER,
  largo_cm INTEGER,
  ancho_cm INTEGER
);

CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email TEXT UNIQUE NOT NULL,
  nombre TEXT,
  auto_id UUID REFERENCES autos(id)
);

CREATE TABLE estacionamientos (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES users(id),
  lat DOUBLE PRECISION NOT NULL,
  lng DOUBLE PRECISION NOT NULL,
  largo_cm INTEGER,
  ancho_cm INTEGER,
  timestamp TIMESTAMPTZ DEFAULT NOW(),
  status TEXT NOT NULL CHECK (status IN ('libre','ocupado'))
);

CREATE TABLE publicidades (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  marca TEXT,
  lat DOUBLE PRECISION,
  lng DOUBLE PRECISION,
  texto TEXT,
  url TEXT,
```

```
  activo BOOLEAN DEFAULT true
);
```

Estos datos pueden manipularse desde Flutter usando el cliente de Supabase. Por ejemplo, para consultar lugares libres: `supabase.from('estacionamientos').select().eq('status','libre')`.

Backend con Supabase

Supabase actúa como BaaS: maneja autenticación, almacenamiento (DB) y funciones serverless. La lógica de negocio incluye:

- **Guardar lugares:** Cuando un usuario marca ocupando un lugar, se inserta en `estacionamientos` con `status = 'ocupado'`. Cuando otro usuario ocupa ese lugar, la app puede actualizar ese registro a "ocupado" también (p.ej. al arrancar su auto en esas coordenadas).
- **Actualizar lugar:** Al confirmar que deja el lugar, se actualiza el registro a `status = 'libre'`.
- **Limpieza automática:** Para evitar congestión de datos (lugares muy antiguos o sin confirmar), se programa un *cron job* usando `pg_cron` en Supabase. Supabase soporta el uso de la extensión `pg_cron` para programar tareas periódicas en la base de datos [9]. Se puede crear una función que borre registros antiguos, por ejemplo:

  ```
  -- Supongamos que los lugares "libres" se borran tras 24 horas
  DELETE FROM estacionamientos
  WHERE status = 'libre' AND timestamp < NOW() - INTERVAL '24 hours';
  ```

  Y programar su ejecución diaria:

  ```
  SELECT cron.schedule('borrar-antiguos', '0 2 * * *', $$
    DELETE FROM estacionamientos
    WHERE status = 'libre' AND timestamp < NOW() - INTERVAL '1 day';
  $$);
  ```

  De este modo, se limpian las “manchas” demasiado viejas sin confirmación.

- **Funciones Edge:** También se pueden implementar Edge Functions (JavaScript/TypeScript) en Supabase si se necesita lógica personalizada (por ejemplo, cálculos complejos o integraciones externas).

Recomendador de Lugares

Parkit sugiere lugares basados en dimensiones. La lógica es simple: dado el auto del usuario (por ejemplo 420 cm x 180 cm), recomendamos cualquier lugar cuyo largo $\geq$ 420 cm y ancho $\geq$ 180 cm (más un pequeño margen, p.ej. +10 cm de seguridad). En SQL sería:

```sql
SELECT * FROM estacionamientos
WHERE status = 'libre'
  AND largo_cm >= 430
  AND ancho_cm >= 190;
```

En Flutter, al mostrar los detalles del pin, se puede consultar la tabla `autos` para obtener los autos del usuario y luego filtrar los `estacionamientos`. Así, el usuario ve sólo lugares donde su auto puede entrar cómodamente.

Tecnologías y Librerías

- **Flutter + Provider/Riverpod:** Para estructura de estado y navegación. `provider` simplifica el acceso a servicios (e.g. Auth, DB) desde cualquier widget. También podría usarse [Riverpod] o Bloc.
- **Supabase Flutter SDK:** Maneja autenticación y CRUD a la base de datos PostgreSQL. Internamente genera consultas HTTP hacia los endpoints de Supabase [5] [6]. Ejemplo de inicialización en Dart ya mostrado.
- **google_maps_flutter:** El plugin oficial de Google Maps para Flutter, usado para integrar el mapa nativo con marcadores, zoom, cámara, etc. Se configuran eventos `onTap`, animaciones de cámara, etc.
- **sensors_plus:** Para detectar movimiento con acelerómetro/giroscopio [1]. Esto permite saber cuándo el usuario arranca el auto.
- **geolocator:** Para obtener la ubicación (lat/lng) del dispositivo [2]. Se usa al compartir lugar (posición inicial) y opcionalmente para calcular proximidad a anuncios o detectar desplazamiento.
- **flutter_local_notifications:** Para enviar notificaciones locales al usuario (p.ej. cuando se detecta que deja el lugar) [3]. Se configura con permisos y canales personalizados.
- **Animated widgets:** Para animaciones de interfaz (expansión de globos, transiciones de página, botones flotantes animados). Flutter facilita animaciones con `AnimatedContainer`, `Hero`, `FadeTransition`, etc.

Extras Opcionales

- **Gamificación:** Se pueden implementar insignias o puntos según la cantidad de lugares compartidos. Por ejemplo, otorgar una medalla “Colaborador” al usuario que compartió 10 plazas. Esto requeriría una tabla extra o campos en `users` para el conteo.
- **Modo oscuro:** Ofrecer un tema oscuro es sencillo ajustando `ThemeData.dark()` paralelo al tema principal. Flutter permite detectar tema del sistema y cambiar dinámicamente.
- **Modo invitado (“solo ver”):** Permitir acceder al mapa sin loguearse, pero deshabilitar acciones críticas (no permitir compartir lugares, solo ver ubicaciones). Al intentar compartir, forzar login. Esto requiere diferenciar la UI según `currentSession == null`.

Código de Ejemplo (Fragments)

A continuación se muestra código ejemplar de partes clave (en Dart, dentro de Flutter):

```dart
// main.dart (inicialización Supabase y Flutter)
import 'package:flutter/material.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'screens/login_screen.dart';
import 'screens/map_screen.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Supabase.initialize(
    url: 'https://<TU-PROYECTO>.supabase.co',
    anonKey: '<TU-ANON-KEY>',
  ); // Inicializa Supabase 4
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Parkit',
      theme: ThemeData(primarySwatch: Colors.lightBlue, fontFamily: 'Roboto'),
      home: AuthGate(),
    );
  }
}

// Widget que decide si mostrar login o mapa según autenticación
class AuthGate extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final session = Supabase.instance.client.auth.currentSession;
    if (session != null) {
      return MapScreen();
    } else {
      return LoginScreen();
    }
  }
}
```

```dart
// login_screen.dart (autenticación)
class LoginScreen extends StatefulWidget { /*...*/ }

class _LoginScreenState extends State<LoginScreen> {
  final emailCtrl = TextEditingController();
  final passCtrl = TextEditingController();
```

```
  void _login() async {
    final res = await Supabase.instance.client.auth.signInWithPassword(
      email: emailCtrl.text.trim(),
      password: passCtrl.text.trim(),
    ); // Loguea usuario 6
    if (res.error != null) {
      // Mostrar error
    } else {
      Navigator.pushReplacement(context, MaterialPageRoute(builder: (_) =>
MapScreen()));
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          // Formularios y botones login/signup
          TextField(controller: emailCtrl, decoration:
InputDecoration(labelText: 'Email')),
          TextField(controller: passCtrl, decoration:
InputDecoration(labelText: 'Contraseña')),
          ElevatedButton(onPressed: _login, child: Text('Iniciar sesión')),
          TextButton(onPressed: () {
            // Navegar a SignupScreen
          }, child: Text('Registrarse')),
        ],
      ),
    );
  }
}
```

```
// map_screen.dart (mapa y marcadores)
class MapScreen extends StatefulWidget { /*...*/ }

class _MapScreenState extends State<MapScreen> {
  GoogleMapController? _mapController;
  Set<Marker> _marcadores = {};

  @override
  void initState() {
    super.initState();
    _cargarMarcadoresLibres();
    _cargarPublicidad();
  }
```

```
  void _cargarMarcadoresLibres() async {
    final userId = Supabase.instance.client.auth.currentUser!.id;
    final res = await Supabase.instance.client
      .from('estacionamientos')
      .select()
      .eq('status', 'libre');
    final lugares = res.data as List;
    setState(() {
      _marcadores = lugares.map((lug) {
        return Marker(
          markerId: MarkerId(lug['id']),
          position: LatLng(lug['lat'], lug['lng']),
          icon:
BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueGreen),
          onTap: () => _mostrarInfoLugar(lug),
        );
      }).toSet();
    });
  }

  void _mostrarInfoLugar(Map lug) {
    final largo = lug['largo_cm'];
    final ancho = lug['ancho_cm'];
    // Filtrar autos recomendados:
    // (Ejemplo: autos del usuario o autos sugeridos por general)
    showModalBottomSheet(context: context, builder: (_) {
      return Column(
        children: [
          Text('Lugar: ${largo}cm x ${ancho}cm'),
          // Listado de autos recomendados...
        ],
      );
    });
  }

  void _cargarPublicidad() async {
    final res = await Supabase.instance.client
      .from('publicidades')
      .select()
      .eq('activo', true);
    final ads = res.data as List;
    setState(() {
      ads.forEach((ad) {
        _marcadores.add(Marker(
          markerId: MarkerId('ad-${ad['id']}'),
          position: LatLng(ad['lat'], ad['lng']),
          icon:
```

```
BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
          onTap: () {
            // Mostrar detalles o navegar a ad['url']
          },
        ));
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: GoogleMap(
        initialCameraPosition: CameraPosition(target: LatLng(-34.61, -58.38),
zoom: 13),
        markers: _marcadores,
        onMapCreated: (c) { _mapController = c; },
        myLocationEnabled: true,
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.local_parking),
        onPressed: _compartirLugarOcupado,
      ),
      floatingActionButtonLocation: FloatingActionButtonLocation.centerDocked,
    );
  }

  void _compartirLugarOcupado() async {
    Position pos = await Geolocator.getCurrentPosition(); // Ubicación actual 2
    final user = Supabase.instance.client.auth.currentUser!;
    final currentCar = /* obtener auto del usuario de DB */;
    await Supabase.instance.client.from('estacionamientos').insert({
      'user_id': user.id,
      'lat': pos.latitude,
      'lng': pos.longitude,
      'largo_cm': currentCar.largo,
      'ancho_cm': currentCar.ancho,
      'status': 'ocupado'
    });
    // Confirmar en UI...
    // Luego, empezar a escuchar acelerómetro para detectar movimiento
    sensorsAccelerometer();
  }

  void sensorsAccelerometer() {
    sensors_plus.SensorsPlatform.instance.accelerometerEvents.listen((event) {
      double mov = event.x.abs() + event.y.abs() + event.z.abs();
      if (mov > 15) {
```

```
        _pedirConfirmacionLugarLibre();
      }
    });
  }

  void _pedirConfirmacionLugarLibre() {
    // Mostrar notificación local [3] y al confirmar:
    // Actualizar estado a 'libre'
    Supabase.instance.client.from('estacionamientos')
      .update({'status': 'libre', 'timestamp':
DateTime.now().toIso8601String()})
      .eq('user_id', Supabase.instance.client.auth.currentUser!.id)
      .eq('status', 'ocupado');
    // Recargar marcadores
    _cargarMarcadoresLibres();
  }
}
```

En este ejemplo se ve cómo se integran las funcionalidades: autenticación Supabase, consulta e inserción en la base, manejo de sensores (`SensorsPlatform.instance.accelerometerEvents`) [1], uso de Google Maps (`GoogleMap` con marcadores) y notificaciones locales.

Guía de Inicio y Ejecución

Para compilar y probar Parkit, seguir estos pasos:

1. **Clonar el repositorio:** Obtener el código fuente y abrir en editor o IDE (Visual Studio Code o Android Studio).
2. **Instalar dependencias:** Ejecutar `flutter pub get` para descargar los paquetes especificados en `pubspec.yaml`.
3. **Configurar Supabase:**
4. Crear un **proyecto en Supabase** y copiar el `URL` y `anonKey`.
5. En el código (`main.dart`), reemplazar `https://<TU-PROYECTO>.supabase.co` y `<TU-ANON-KEY>` por los valores reales.
6. En el dashboard de Supabase, importar las tablas (`users`, `autos`, `estacionamientos`, `publicidades`) según el esquema mostrado.
7. (Opcional) Habilitar políticas RLS si se desea seguridad adicional.
8. **Configurar Google Maps:**
9. Obtener **API Key** para Android/iOS desde Google Cloud Console.
10. Añadir la clave en AndroidManifest.xml y en Info.plist.
11. **Permisos móviles:** Asegurarse de declarar permisos de ubicación y sensores en `AndroidManifest.xml` y las descripciones necesarias en `Info.plist` (por ejemplo, `NSLocationWhenInUseUsageDescription` y `NSMotionUsageDescription`).
12. **Ejecutar la app:** Conectar dispositivo/emulador y correr `flutter run`. Debería compilar sin errores y mostrar la pantalla de login.

13. **Probar funcionalidades:** Crear una cuenta, agregar datos de auto, compartir un lugar en el mapa y verificar que aparece para otros usuarios. Probar que al salir del lugar (simulando movimiento) la app pide confirmación y vuelve a mostrar el pin.

Con esto, Parkit queda listo para un desarrollo real, integrando frontend en Flutter y backend en Supabase, con capacidades de geolocalización, notificaciones y monetización embebida. Este diseño es escalable: por ejemplo se pueden agregar estadísticas de uso, filtros de búsqueda, o integración con pasarelas de pago en el futuro.

Fuentes: Las soluciones técnicas aquí descritas se basan en documentación oficial de Supabase (registro y login con email/contraseña [5] [6]), Google Maps para Flutter, y referencias de Flutter sobre sensores y notificaciones [1] [3] [2] [8] [7], entre otras. Estos recursos confirman la viabilidad de cada componente en la app.

---

[1] sensors_plus | Flutter package
https://pub.dev/packages/sensors_plus

[2] geolocator | Flutter package
https://pub.dev/packages/geolocator

[3] flutter_local_notifications | Flutter package
https://pub.dev/packages/flutter_local_notifications

[4] Flutter: Initializing | Supabase Docs
https://supabase.com/docs/reference/dart/initializing

[5] Flutter: Create a new user | Supabase Docs
https://supabase.com/docs/reference/dart/auth-signup

[6] Flutter: Sign in a user | Supabase Docs
https://supabase.com/docs/reference/dart/auth-signinwithpassword

[7] Add A Custom Info Window to your Google Map Pins in Flutter | by Roman Jaquez | Flutter Community | Medium
https://medium.com/flutter-community/add-a-custom-info-window-to-your-google-map-pins-in-flutter-2e96fdca211a

[8] Creating Custom Markers in Flutter: A Comprehensive Guide (Widget to custom marker). | by Pvaddoriya | Medium
https://medium.com/@pvaddoriya1246/creating-custom-markers-in-flutter-a-comprehensive-guide-widget-to-custom-marker-9f2bef3fa614

[9] Scheduling Edge Functions | Supabase Docs
https://supabase.com/docs/guides/functions/schedule-functions