RideMatch — Full-stack Starter (Flutter + Firebase)

> Minimal, runnable starter project that demonstrates a ride-comparison & booking flow. All data is mocked so you can test without real provider APIs.

Repo tree (zip-able)

```
RideMatch/
├── README.md
├── .env.example
├── firebase.json
├── firestore.rules
├── functions/
│   ├── Dockerfile
│   ├── package.json
│   ├── index.js        <-- Cloud Functions (Express) - compare & book
│   └── .env.example
├── hosting/
│   └── admin/
│       ├── index.html  <-- static admin page listing bookings
│       └── admin.js
└── mobile/
    ├── pubspec.yaml
    └── lib/
        ├── main.dart
        ├── services/
        │   └── api.dart  <-- calls functions endpoints
        └── screens/
            ├── splash.dart
            ├── login_otp.dart
            ├── home_map.dart
            ├── compare.dart
            ├── booking.dart
            ├── tracking.dart
            ├── history.dart
            └── settings.dart
```

Quick description

- **Mobile**: Flutter app (Dart) with screens: Splash, Login (OTP mock), Home (Google Map), Compare, Booking, Tracking (mock), History, Settings.
- **Map**: `google_maps_flutter` with placeholder `GOOGLE_MAPS_API_KEY` (show how to add it below).
- **Backend**: Firebase Cloud Functions (Node.js, Express)
- `POST /api/compare` — accepts `{ pickup:{lat,lng}, drop:{lat,lng} }` and returns mocked providers/offers.
- `POST /api/book` — accepts `{ userId, provider, pickup, drop, fare }`, writes to Firestore `bookings` collection and returns `bookingId`.
- **Firestore**: example schemas for `users` and `bookings` provided later.
- **Admin**: static HTML served via Firebase Hosting that lists bookings and shows total commission (15%).
- **Notes**: Everything is mocked so you can test locally and with Firebase emulator. Replace API keys only when you're ready.

File: `functions/index.js` (sample Cloud Functions)

```
// functions/index.js
const functions = require('firebase-functions');
const admin = require('firebase-admin');
const express = require('express');
const cors = require('cors');

admin.initializeApp();
const db = admin.firestore();

const app = express();
app.use(cors({ origin: true }));
app.use(express.json());

// Helper: mocked providers
function mockedOffers(pickup, drop) {
  // Simple mocked data: prices scale with distance approx.
  const distFactor = Math.max(1, Math.abs(pickup.lat - drop.lat) +
Math.abs(pickup.lng - drop.lng));
  const base = Math.round(100 * distFactor);
  const now = Date.now();
  return [
    { provider: 'uber', price: Math.round(base * 1.8), eta: 3 +
Math.round(Math.random() * 5), vehicle: 'Car' },
    { provider: 'lyft', price: Math.round(base * 1.7), eta: 2 +
Math.round(Math.random() * 6), vehicle: 'Car' },
    { provider: 'bolt', price: Math.round(base * 1.5), eta: 4 +
```

```
Math.round(Math.random() * 4), vehicle: 'Car' },
    { provider: 'bikeX', price: Math.round(base * 0.9), eta: 6 +
Math.round(Math.random() * 8), vehicle: 'Bike' }
  ];
}

// POST /api/compare
app.post('/api/compare', async (req, res) => {
  try {
    const { pickup, drop } = req.body;
    if (!pickup || !drop) return res.status(400).json({ error: 'pickup and drop
required' });

    const offers = mockedOffers(pickup, drop);
    return res.json({ offers });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'internal' });
  }
});

// POST /api/book
app.post('/api/book', async (req, res) => {
  try {
    const { userId, provider, pickup, drop, fare } = req.body;
    if (!userId || !provider || !pickup || !drop || fare == null) {
      return res.status(400).json({ error: 'missing fields' });
    }

    const booking = {
      userId,
      provider,
      pickup,
      drop,
      fare,
      status: 'confirmed',
      createdAt: admin.firestore.FieldValue.serverTimestamp()
    };

    const docRef = await db.collection('bookings').add(booking);
    return res.json({ bookingId: docRef.id });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'internal' });
  }
});

exports.api = functions.https.onRequest(app);
```

Save the file as `functions/index.js`. This exposes endpoints at `https://<region>-<project>.cloudfunctions.net/api/compare` and `/api/book` (or under emulator URL).

`functions/package.json`

```
{
  "name": "functions",
  "version": "1.0.0",
  "main": "index.js",
  "engines": { "node": "18" },
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "firebase-admin": "^11.0.0",
    "firebase-functions": "^4.0.0"
  }
}
```

`functions/Dockerfile`

```
FROM node:18-alpine
WORKDIR /usr/src/app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
CMD ["node", "index.js"]
```

This Dockerfile is optional — useful for reproducible builds or container-based deployments.

Firebase Hosting Admin page (`hosting/admin/index.html`)

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>RideMatch Admin</title>
  <script src="https://www.gstatic.com/firebasejs/9.22.0/firebase-app-compat.js"></script>
```

```
  <script src="https://www.gstatic.com/firebasejs/9.22.0/firebase-firestore-compat.js"></script>
  <meta name="viewport" content="width=device-width,initial-scale=1">
</head>
<body>
  <h1>RideMatch — Admin</h1>
  <div>Total Bookings: <span id="count">0</span></div>
  <div>Total Commission (15%): <span id="commission">0</span></div>
  <ul id="list"></ul>

  <script>
    // TODO: replace with your app's Firebase config (for quick local testing you can keep it here)
    const firebaseConfig = {
      apiKey: "REPLACE_ME",
      authDomain: "REPLACE_ME",
      projectId: "REPLACE_ME"
    };
    firebase.initializeApp(firebaseConfig);
    const db = firebase.firestore();

    async function load() {
      const snapshot = await db.collection('bookings').orderBy('createdAt','desc').get();
      const list = document.getElementById('list');
      let total = 0;
      list.innerHTML = '';
      snapshot.forEach(doc => {
        const b = doc.data();
        total += (b.fare || 0);
        const li = document.createElement('li');
        li.textContent = `${doc.id} — ${b.userId} — ${b.provider} — fare: ${b.fare} — status: ${b.status}`;
        list.appendChild(li);
      });
      document.getElementById('count').textContent = snapshot.size;
      document.getElementById('commission').textContent = (total * 0.15).toFixed(2);
    }

    load();
  </script>
</body>
</html>
```

Flutter mobile — key files

`mobile/pubspec.yaml` **(minimal)**

```
name: ridematch
description: RideMatch starter
publish_to: 'none'
version: 0.0.1

environment:
  sdk: '>=2.18.0 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  google_maps_flutter: ^2.2.0
  http: ^0.13.4
  flutter_dotenv: ^5.0.2

flutter:
  uses-material-design: true
```

`mobile/lib/main.dart` **(entry + routing)**

```
import 'package:flutter/material.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'screens/splash.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await dotenv.load(fileName: ".env");
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'RideMatch',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: const SplashScreen(),
    );
```

```
  }
}
```

`mobile/lib/screens/splash.dart`

```
import 'dart:async';
import 'package:flutter/material.dart';
import 'login_otp.dart';

class SplashScreen extends StatefulWidget {
  const SplashScreen({Key? key}) : super(key: key);

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    super.initState();
    Timer(const Duration(seconds: 2), () {
      Navigator.pushReplacement(context, MaterialPageRoute(builder: (_) =>
const LoginOtp()));
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(child: Text('RideMatch', style: TextStyle(fontSize: 32,
fontWeight: FontWeight.bold))),
    );
  }
}
```

`mobile/lib/screens/login_otp.dart` **(mock OTP flow)**

```
import 'package:flutter/material.dart';
import 'home_map.dart';

class LoginOtp extends StatefulWidget {
  const LoginOtp({Key? key}) : super(key: key);
  @override
  State<LoginOtp> createState() => _LoginOtpState();
}
```

```
class _LoginOtpState extends State<LoginOtp> {
  final _phoneCtrl = TextEditingController();
  final _codeCtrl = TextEditingController();
  bool _codeSent = false;

  void _sendOtp() {
    // Mock: just flip state, don't call Firebase.
    setState(() => _codeSent = true);
  }

  void _verifyOtp() {
    // Mock success
    Navigator.pushReplacement(context, MaterialPageRoute(builder: (_) => const
HomeMap()));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Login (Mock OTP)')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(children: [
          TextField(controller: _phoneCtrl, decoration:
InputDecoration(labelText: 'Phone')),
          SizedBox(height: 12),
          if (!_codeSent)
            ElevatedButton(onPressed: _sendOtp, child: Text('Send OTP (mock)'))
          else ...[
            TextField(controller: _codeCtrl, decoration:
InputDecoration(labelText: 'Enter OTP (any)')),
            ElevatedButton(onPressed: _verifyOtp, child: Text('Verify'))
          ]
        ]),
      ),
    );
  }
}
```

`mobile/lib/screens/home_map.dart` **(Google Map placeholder)**

```
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'compare.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
```

```
class HomeMap extends StatefulWidget {
  const HomeMap({Key? key}) : super(key: key);

  @override
  State<HomeMap> createState() => _HomeMapState();
}

class _HomeMapState extends State<HomeMap> {
  late GoogleMapController mapController;
  final LatLng _center = const LatLng(37.7749, -122.4194);

  @override
  Widget build(BuildContext context) {
    // Note: set your GOOGLE_MAPS_API_KEY in platform-specific config.
    return Scaffold(
      appBar: AppBar(title: Text('Home')),
      body: GoogleMap(
        onMapCreated: (controller) => mapController = controller,
        initialCameraPosition: CameraPosition(target: _center, zoom: 12.0),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.search),
        onPressed: () => Navigator.push(context, MaterialPageRoute(builder: (_)
=> CompareScreen())),
      ),
    );
  }
}
```

`mobile/lib/screens/compare.dart` **(calls /api/compare)**

```
import 'package:flutter/material.dart';
import '../services/api.dart';

class CompareScreen extends StatefulWidget {
  const CompareScreen({Key? key}) : super(key: key);
  @override
  State<CompareScreen> createState() => _CompareScreenState();
}

class _CompareScreenState extends State<CompareScreen> {
  bool loading = false;
  List offers = [];

  void _compare() async {
```

```
    setState(() => loading = true);
    final res = await Api.compare({ 'pickup': { 'lat': 37.7749, 'lng':
-122.4194 }, 'drop': { 'lat': 37.7849, 'lng': -122.4094 } });
    setState(() { offers = res['offers'] ?? []; loading = false; });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Compare')),
      body: Column(children: [
        ElevatedButton(onPressed: _compare, child: Text('Get Offers (mock)')),
        if (loading) CircularProgressIndicator(),
        Expanded(
          child: ListView.builder(
            itemCount: offers.length,
            itemBuilder: (_, i) {
              final o = offers[i];
              return ListTile(
                title: Text('${o['provider']} — ${o['vehicle']}'),
                subtitle: Text('Fare: ${o['price']} | ETA: ${o['eta']} min'),
                trailing: ElevatedButton(onPressed: () {
                  Navigator.push(context, MaterialPageRoute(builder: (_) =>
BookingScreen(offer: o)));
                }, child: Text('Book')),
              );
            }
          ),
        )
      ]),
    );
  }
}
```

`mobile/lib/screens/booking.dart` **(calls /api/book)**

```
import 'package:flutter/material.dart';
import '../services/api.dart';

class BookingScreen extends StatefulWidget {
  final Map offer;
  BookingScreen({required this.offer});
  @override
  State<BookingScreen> createState() => _BookingScreenState();
}
```

```
class _BookingScreenState extends State<BookingScreen> {
  bool booking = false;
  String? bookingId;

  void _book() async {
    setState(() => booking = true);
    final res = await Api.book({
      'userId': 'mock-user-1',
      'provider': widget.offer['provider'],
      'pickup': { 'lat': 37.7749, 'lng': -122.4194 },
      'drop': { 'lat': 37.7849, 'lng': -122.4094 },
      'fare': widget.offer['price']
    });
    setState(() { bookingId = res['bookingId']; booking = false; });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Booking')),
      body: Center(
        child: booking ? CircularProgressIndicator() : Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text('Provider: ${widget.offer['provider']}'),
            Text('Fare: ${widget.offer['price']}'),
            ElevatedButton(onPressed: _book, child: Text('Confirm Booking')),
            if (bookingId != null) Text('Booked! id: $bookingId')
          ],
        ),
      ),
    );
  }
}
```

`mobile/lib/services/api.dart`

```
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:flutter_dotenv/flutter_dotenv.dart';

class Api {
  static final String base = dotenv.env['FUNCTIONS_BASE_URL'] ?? 'http://
localhost:5001/your-project/us-central1/api';

  static Future<Map> compare(Map body) async {
```

```
    final r = await http.post(Uri.parse('$base/api/compare'), body:
jsonEncode(body), headers: { 'Content-Type': 'application/json' });
    return jsonDecode(r.body) as Map;
  }

  static Future<Map> book(Map body) async {
    final r = await http.post(Uri.parse('$base/api/book'), body:
jsonEncode(body), headers: { 'Content-Type': 'application/json' });
    return jsonDecode(r.body) as Map;
  }
}
```

Update `FUNCTIONS_BASE_URL` in `mobile/.env` to point to your emulator or deployed function endpoint.

Firestore rules (`firestore.rules`)

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{uid} {
      allow read, write: if request.auth != null && request.auth.uid == uid;
    }

    match /bookings/{bid} {
      // Allow writes from functions service account, and allow reads for
authenticated users
      allow create: if request.auth != null; // for emulator simplicity
      allow read: if request.auth != null;
      allow update, delete: if false;
    }

    // Admin-only area could be restricted by project IAM in production
  }
}
```

`.env.example` (root and functions)

```
# mobile/.env
FUNCTIONS_BASE_URL=http://localhost:5001/your-project/us-central1/api
GOOGLE_MAPS_API_KEY=REPLACE_WITH_YOUR_GOOGLE_MAPS_API_KEY
```

```
# functions/.env.example (for local usage if needed)
# If using Firebase emulator, functions will use application default
credentials.
# Example: any env you need for production
NODE_ENV=development
```

`firebase.json` (minimal)

```
{
  "functions": {
    "source": "functions"
  },
  "hosting": [
    {
      "target": "admin",
      "public": "hosting/admin",
      "ignore": ["firebase.json", "**/.*", "**/node_modules/**"]
    }
  ]
}
```

We'll create a hosting target `admin` and deploy the `hosting/admin` folder there.

Firestore example schemas

users (collection)

```
users/{uid} = {
  displayName: string,
  phone: string,
  createdAt: timestamp
}
```

bookings (collection)

```
bookings/{bookingId} = {
  userId: string,
  provider: string,
  pickup: { lat: number, lng: number },
  drop: { lat: number, lng: number },
```

```
  fare: number,
  status: string, // e.g. confirmed, completed, cancelled
  createdAt: timestamp
}
```

README — Exact setup steps

```
# RideMatch — Starter

This repository contains a Flutter mobile app and Firebase Cloud Functions +
Hosting example for a mocked ride comparison / booking flow.

## Prerequisites
- Flutter SDK
- Node.js 18+
- Firebase CLI (`npm install -g firebase-tools`)
- (Optional) Docker

## 1) Create Firebase project
1. Go to https://console.firebase.google.com and create a new project (e.g.
`ridematch-xyz`).
2. Enable **Authentication -> Sign-in method -> Phone** (for real phone auth).
For local mock testing the app uses a mocked flow; to use real phone OTP enable
this.
3. Enable **Firestore** (in Native mode).
4. Enable **Cloud Functions** (Node 18 recommended).
5. Set up **Hosting** for the admin static page.

## 2) Clone repository
```bash
git clone <this-repo> RideMatch
cd RideMatch
```

3) Configure functions

1. `cd functions`
2. Copy `.env.example` to `.env` if you need environment variables for functions (emulator typically uses app default creds).
3. `npm install`

4) Configure mobile

1. `cd mobile`
```

2. Copy `.env.example` to `.env` and edit `FUNCTIONS_BASE_URL` to point to your functions emulator or deployed URL.
3. Example emulator URL: `http://localhost:5001/<your-project>/us-central1/api`
4. Add your `GOOGLE_MAPS_API_KEY` in the platform-specific config:
5. **Android**: open `android/app/src/main/AndroidManifest.xml` and add inside `<application>`:

```
<meta-data android:name="com.google.android.geo.API_KEY"
android:value="REPLACE_WITH_KEY"/>
```

6. **iOS**: set API key in `AppDelegate` per `google_maps_flutter` docs.
7. `flutter pub get`

5) Run with Firebase emulator (recommended for dev)

1. From project root, run:

```
firebase emulators:start --only functions,firestore,hosting
```

2. This starts Functions on `http://localhost:5001` — update `mobile/.env` `FUNCTIONS_BASE_URL` accordingly.
3. Run the Flutter app on emulator or device:

```
cd mobile
flutter run
```

6) Deploy to Firebase

1. Login: `firebase login`
2. Initialize (if not already): `firebase init` and choose Functions + Hosting + Firestore rules.
3. For Hosting, set `hosting/admin` as public for the `admin` target.
4. Deploy:

```
firebase deploy --only functions,hosting
```

5. After deploy, your functions base URL will look like `https://us-central1-<project>.cloudfunctions.net/api` — update `mobile/.env` `FUNCTIONS_BASE_URL` to that value and rebuild the app.

7) Build release APK (Android)

From `mobile/`:

```
flutter build apk --release
```

The generated APK will be in `build/app/outputs/flutter-apk/app-release.apk`.

8) Admin page

The admin page is served at your hosting URL (for the hosting target). It reads bookings from Firestore and shows a 15% commission total.

Notes

- All provider offers are mocked inside `functions/index.js` — replace with real provider integrations when available.
- Use Firebase Emulator for local testing to avoid costs and speed up iteration.
- For production, secure Firestore rules and restrict hosting access (admin should be authenticated + role-based). ```

Additional tips & comments

- The mobile app uses a **mock OTP** flow for convenience — if you want real Phone Auth, integrate `firebase_auth` and follow the Firebase docs.
- The `compare` endpoint returns deterministic-ish mocked provider offers; adjust `mockedOffers` if you want to simulate surge pricing.
- Commission on admin is computed client-side in the provided admin page as `total * 0.15`.

If you'd like, I can also: - Generate the exact Flutter project files for download (zipped) — tell me if you want the full file contents in this canvas. - Expand the admin UI to allow status updates (complete/cancel) and CSV export.

Enjoy building RideMatch! 🚗