**Signup with Google:**

**🚧 Overview**

* **Frontend**: Flutter with google\_sign\_in package
* **Backend**: Node.js (Express), verifying Google ID token
* **Auth Flow**:
  1. User signs in with Google on Flutter.
  2. Flutter gets an **ID token** from Google.
  3. Flutter sends that token to your Node.js backend.
  4. Node.js verifies the token with Google and logs in / registers the user.

## ✅ Step 1: Set up Google API (OAuth 2.0 Client ID)

1. Go to Google Cloud Console.
2. Create/select a project.
3. Navigate to **APIs & Services > Credentials**.
4. Create **OAuth 2.0 Client ID**:
   * Application type: **Android** (and optionally Web for development).
   * Save the **Client ID** — needed for both frontend and backend.

## ✅ Step 2: Flutter Frontend

### 🔌 Add Dependencies

In pubspec.yaml:

yaml

CopyEdit

dependencies:

google\_sign\_in: ^6.1.0

http: ^0.13.6

### 🔐 Google Sign-In and Token Sending

dart

CopyEdit

import 'package:google\_sign\_in/google\_sign\_in.dart';

import 'package:http/http.dart' as http;

final GoogleSignIn \_googleSignIn = GoogleSignIn(

scopes: ['email'],

);

Future<void> signInWithGoogle() async {

try {

final GoogleSignInAccount? account = await \_googleSignIn.signIn();

final GoogleSignInAuthentication auth = await account!.authentication;

final idToken = auth.idToken;

// Send token to backend

final response = await http.post(

Uri.parse('https://your-backend.com/api/auth/google'),

headers: {'Content-Type': 'application/json'},

body: '{"idToken": "$idToken"}',

);

if (response.statusCode == 200) {

print('Signed in successfully: ${response.body}');

} else {

print('Sign-in failed: ${response.body}');

}

} catch (e) {

print('Google sign-in error: $e');

}

}

## ✅ Step 3: Node.js Backend

### 🧱 Install packages

bash

CopyEdit

npm install express google-auth-library jsonwebtoken dotenv cors

### 📂 Folder structure (simple)

bash

CopyEdit

project/

├── app.js

├── .env

└── routes/

└── auth.js

### ✍️ .env

ini

CopyEdit

GOOGLE\_CLIENT\_ID=your-client-id.apps.googleusercontent.com

JWT\_SECRET=your\_jwt\_secret\_key

### 📁 routes/auth.js

js

CopyEdit

const express = require('express');

const router = express.Router();

const { OAuth2Client } = require('google-auth-library');

const jwt = require('jsonwebtoken');

const client = new OAuth2Client(process.env.GOOGLE\_CLIENT\_ID);

router.post('/google', async (req, res) => {

const { idToken } = req.body;

try {

const ticket = await client.verifyIdToken({

idToken,

audience: process.env.GOOGLE\_CLIENT\_ID,

});

const payload = ticket.getPayload();

// Get user data

const { sub, email, name, picture } = payload;

// You can store/retrieve the user from DB here

// For now, just create a token and send back

const token = jwt.sign({ id: sub, email }, process.env.JWT\_SECRET, {

expiresIn: '7d',

});

res.status(200).json({ token, user: { email, name, picture } });

} catch (error) {

console.error('Google Sign-In Error:', error);

res.status(401).json({ error: 'Invalid Google token' });

}

});

module.exports = router;

### 📁 app.js

js

CopyEdit

const express = require('express');

const dotenv = require('dotenv');

const cors = require('cors');

dotenv.config();

const app = express();

app.use(cors());

app.use(express.json());

const authRoutes = require('./routes/auth');

app.use('/api/auth', authRoutes);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

## 🧪 Testing

* Run your backend: node app.js
* Run your Flutter app
* Tap "Sign in with Google"
* Backend should verify token and return a custom JWT

SHA key:  
fcfb9b054a72727da8c82ec81c3037384c58cbef

Features Included:

Full-Text Search (search parameter)

Price Filtering (using operators):

rate[gte]=100 (Minimum price)

rate[lte]=500 (Maximum price)

Availability Filter (availability=true|false)

Sorting (sort=price,-createdAt)

Field Selection (fields=title,price,availability)

Pagination (page=2&limit=10)

Date Filtering:

createdAfter=2024-01-01

createdBefore=2024-12-31

Combined Filters (All parameters can be used together)

Example Requests:

Basic search with pagination:

bash

Copy

GET /api/v1/services?search=web&page=2&limit=10

Price range with availability:

bash

Copy

GET /api/v1/services?rate[gte]=100&rate[lte]=500&availability=true

Advanced filtering with sorting:

bash

Copy

GET /api/v1/services?search=design&rate[gte]=200&sort=-rate,createdAt&fields=title,rate

Date filtering:

bash

Copy

GET /api/v1/services?createdAfter=2024-01-01&createdBefore=2024-12-31