
📱🧠 AutoText – Smart Social Media Caption Generator

> “Upload a photo. Let AI generate the perfect caption. Choose your tone. Copy, share, or save — all in one click.”

✅ 1. Problem Statement (What It Solves)

❓ Problem:

People often struggle to write creative, mood-matching captions for their social media posts.

💡 Solution:

Use AI (Vision + Language) to:

- Analyze an image
- Understand context (person, mood, setting)
- Let user pick a tone (e.g., funny, romantic, poetic)
- Generate catchy, context-aware captions instantly
- Let users copy, share, or save captions

🔨 2. System Architecture Overview

```
[Flutter Mobile]      [React Web]
      |                     |
      | --> REST API --> [Python AI Backend - Flask/FastAPI]
                            |
              ┌─────────────┼─────────────┐
       [Caption Generator] [Vision Analysis] [Tone Handler]
                            |
                [Firebase Firestore DB]
                            |
                [Firebase Cloud Storage]
                            |
           [Firebase Cloud Messaging (FCM)]
                            |
               [Razorpay / Stripe for Payments]
```

📲 3. Mobile & Web Apps (Flutter & React)

🧩 Shared Core Features (Both platforms):

- User login/signup (Firebase)
- Upload or capture image
- Select tone → request captions from AI backend
- View and copy captions
- View caption history
- Buy premium plan
- Notifications and reminders

📱 Flutter App (Mobile):

- Camera access
- Biometric login (optional)
- Push notifications (via FCM)

- Offline history caching (Hive/LocalStorage)

💻 React Web App:

- Responsive layout
- Drag & drop image upload
- Admin dashboard (optional)
- Showcase portfolio work (great for demos)

🎨 4. UI Pages Overview

Page	Mobile	Web
Splash & Onboarding	✅	❌
Login/Signup	✅	✅
Home (Upload + Tone)	✅	✅
Captions Results	✅	✅
History	✅	✅
Premium Plans	✅	✅
Profile / Settings	✅	✅
Admin Panel	❌	✅
About Us	✅	✅

🖼️ 5. Icons, Images, & UI Design Guide

- 🎨 **Font**: Poppins or Inter
- 🖼️ **Images**: Custom illustrations (for onboarding), user-uploaded photos
- 📦 **Icons**: `Feather Icons`, `FontAwesome`, or `Material Icons`

- 🧩 **Tone Icons**:
  - 😂 Funny → 😆
  - ❤️ Romantic → 💕
  - ✨ Poetic → ✍️
  - 🔥 Savage → 🔪
  - 🙃 Sarcastic → 👀

💬 6. Notifications

Type	Trigger	Example
Inactivity	3/6/9/15/30 days	“Bhul toh nahi gaye?”
Daily Quote	11AM	“Your audience deserves better captions 👀”
Limit Reached	After 3 free captions	“You’ve hit the daily cap 😢”
Premium Push	Weekly	“Unlock savage mode 🔥 with Premium”

Sent via FCM using backend logic or Firebase Scheduled Functions

---

🧠 7. AI/ML Models to Use

a. 🔍 Image Analysis (Vision AI):

- **Google Cloud Vision API** or **CLIP (OpenAI)** to extract:
  - Tags (beach, dog, food, selfie)
  - Mood (smiles, colors, people count)
- Use in Python to generate caption prompt.

b. 📝 Caption Generator (Text AI):

- **OpenAI GPT-4 / GPT-3.5 Turbo API**
    - Input prompt:
      "Write a funny Instagram caption for a picture showing a couple on a beach during sunset"
- Fine-tuned tone formatting with:
    - Prompt engineering + temperature adjustment
    - Optional: Local LLM using Hugging Face if no API budget

---

⚙️ 8. Backend Setup (Python + Firebase + Payments)

📂 Folder Structure:

```
backend/
 ├ main.py (FastAPI/Flask app)
 ├ /routes/
 |   └ captions.py, auth.py
 ├ /services/
 |   └ gpt_generator.py, vision_analyzer.py
 ├ /utils/
 |   └ auth_utils.py, tone_mapper.py
 └ /models/
    └ user.py, history.py
```

🔌 APIs:

- `POST /generate_caption` → Upload image + tone → returns captions
- `POST /save_caption` → Save caption to history (Firestore)
- `GET /history` → Retrieve user's captions
- `POST /notify` → Trigger custom notification via FCM
- `POST /check_subscription` → Verify plan

💰 9. Payments

- **Razorpay** (India) or **Stripe** (Global)
- **Client Integration**:
  - Flutter: `razorpay_flutter`
  - React: Razorpay SDK
- **Backend**:
  - Listen to `payment_success` → mark user as Premium in Firestore

📦 10. Backend-First Workflow (Recommended)

✅ Step-by-Step Order:

1. **Build Python backend (FastAPI or Flask)**:
   - Test image analysis and caption generator
   - Set up routes + Firebase DB connection
   - Store history properly
2. **Set up Firebase (Firestore + Storage + Auth)**
3. **Integrate payment webhook logic**
4. **Create Flutter screens**
   - First: Login → Home → Generate → Results
5. **React Web next**
   - Use same API as Flutter
6. **Add Notifications last**
   - Cloud Functions or Scheduled Cron Jobs

⏸️ **Stopping here for now to avoid overload.**
👉 Next part will include:

- Firebase DB structure
- Firestore security rules
- Figma layout for screens
- Prompt examples for tone-based captions
- Local data caching
- Deployment options

Awesome! Let's continue with **Part 2** of your full project plan for **AutoText – Smart Social Media Caption Generator**. This part will cover:

1. 🔐 Firebase Database Structure
2. 🛡️ Firestore Security Rules
3. 🎨 Figma Layout Tips (for each screen)
4. ✍️ AI Prompt Templates (tone-based captioning)
5. 💾 Local Data Caching Strategy
6. ☁️ Deployment Plan (Mobile, Web, Backend)
7. 📈 Future Features & Scalability Plan

🔐 1. Firebase Firestore Database Structure

Structure:

Users (Collection)

└ userId (Document)

├ name: string

├ email: string

```
   ├ isPremium: bool

   ├ lastActive: timestamp

   ├ deviceToken: string (for FCM)

   ├ captions (Subcollection)

     └ captionId (Document)

       ├ imageUrl: string

       ├ tone: string

       ├ captions: [array]

       ├ timestamp: datetime
```

Firebase Storage (for images):

/captions/userId/filename.jpg

🛡️ 2. Firestore Security Rules

```
rules_version = '2';

service cloud.firestore {

  match /databases/{database}/documents {



    match /Users/{userId} {

      allow read, write: if request.auth.uid == userId;



      match /captions/{captionId} {

        allow read, write: if request.auth.uid == userId;

      }

    }
    match /AdminData/{doc} {
      allow read, write: if request.auth.token.admin == true;
    }
  }
}
```

Firebase Storage Rules:

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /captions/{userId}/{allPaths=**} {
      allow read, write: if request.auth.uid == userId;
    }
  }
}
```

🎨 3. Figma Layout Tips

Each screen should be designed in a **Mobile (iPhone 13/Android 1080px)** frame with:

Element Type	Use This in Figma
Buttons	Auto Layout → Rounded corners (12–16px)

Image Upload Area	Dotted border + Upload icon
Tone Selection	Scrollable chip row (use icons + emojis)
Captions List	Cards with soft shadows
Icons	Feather Icons or Hero Icons plugin
Fonts	Poppins (Headlines), Inter (Body)
Color Style Guide	Primary, Accent, Light, Dark shades

Use Figma components and variants for buttons, caption cards, etc., so reuse is easier in dev.

✍️ 4. AI Prompt Templates

Your backend should build **custom prompts** based on:

- Tone selected
- Detected image tags or objects
- User preferences

🧠 Prompt Template:

Write a [tone] Instagram caption for a photo that shows [tags from image]. Keep it short, creative, and suitable for social media.

🧾 Examples:

Tone	Prompt	Output
Funny	“Write a funny caption for a group of friends at a beach with sunglasses”	“Squad goals, sun edition 😎☀️”
Romantic	“Romantic caption for a couple holding hands at sunset”	“Together is a wonderful place to be 🌅❤️”
Poetic	“Poetic caption for a foggy morning in the mountains”	“Whispers of clouds in nature’s softest voice”
Savage	“Savage caption for a solo mirror selfie”	“I’m not everyone’s cup of tea. I’m champagne.”

💾 5. Local Data Caching

Mobile (Flutter):

Use `Hive` or `SharedPreferences` to:

- Store last 10 captions offline
- Cache image URLs + tones locally
- Reduce Firestore reads

```
Box captionBox = Hive.box('captions');

captionBox.put('last_caption', captionData);
```

Web (React):

- Use `localStorage` or `IndexedDB` via libraries like `idb-keyval`

☁️ 6. Deployment Plan

🔁 Deployment Order (Backend First):

1. ✅ Build and deploy **Python Backend** (FastAPI) on:
    - Railway / Render / DigitalOcean / Vercel Serverless Functions
    - Use a `.env` for GPT/Google Vision keys
2. ✅ Set up **Firebase** project:
    - Firestore
    - Storage
    - FCM
    - Auth
3. ✅ Integrate backend API in **Flutter app**
4. ✅ Deploy **Flutter app to Play Store**
5. ✅ Build and host **React web**:
    - Netlify / Vercel (connected to GitHub repo)
6. ✅ Add custom domain and SEO meta tags for React

📈 7. Future Features & Scalability

Feature	Why It's Useful
🔍 Hashtag AI	Add relevant hashtags using NLP
📸 Caption over Image	Create sharable caption+image cards

🌐 Multi-language Captions	Expand global reach
📅 Auto-post scheduling	Post on Instagram using 3rd-party API
🛡️ Content moderation	Detect offensive inputs
🧠 Local model	Save cost on GPT-API with LLMs like Mistral/phi-3

🧠 Bonus: Team Roles (for real-world deployment)

Role	Tasks
🧑‍💻 Backend Developer	Python AI, API, Firestore
👨‍🎨 UI/UX Designer	Figma, assets, icons
🧑‍💻 Flutter Dev	Camera, uploads, design
💻 React Dev	Web frontend
📢 Growth Marketer	Push notifications, tone ideas

Perfect. Here's your complete **file/folder structure** for all 3 parts of the AutoText project — Flutter app, Python backend, and React web — with **detailed folder roles, filenames, and responsibilities**.

At the end, I'll give you a **realistic time plan (4-week roadmap)** to build the full product solo or with a small team.

📁 FLUTTER FILE STRUCTURE (Mobile App)

```
lib/
├── main.dart              ← Entry point (initializes Firebase, routes)
├── app.dart               ← MaterialApp + global theme + router
├── routes/                ← Named routes and route config
│   └── app_routes.dart
│
├── core/                  ← Reusable, global services or helpers
│   ├── constants.dart
│   ├── firebase_service.dart
│   ├── notification_service.dart
│   └── auth_service.dart
│
├── data/                  ← Models and Firebase/Storage wrappers
│   ├── models/
│   │   ├── caption_model.dart
│   │   └── user_model.dart
│   └── repositories/
│       ├── caption_repo.dart
│       ├── auth_repo.dart
│       └── history_repo.dart
│
├── ui/                     ← UI layer (screens + widgets)
│   ├── screens/
│   │   ├── splash_screen.dart
│   │   ├── onboarding_screen.dart
│   │   ├── login_screen.dart
│   │   ├── home_screen.dart
│   │   ├── caption_result_screen.dart
│   │   ├── history_screen.dart
│   │   ├── premium_screen.dart
│   │   ├── settings_screen.dart
│   │   └── about_screen.dart
│   └── widgets/
│       ├── tone_selector.dart
│       ├── caption_card.dart
│       ├── image_upload_tile.dart
│       └── premium_plan_card.dart
│
├── services/               ← Camera, biometrics, payments
│   ├── camera_service.dart
│   ├── payment_service.dart
│   ├── biometric_service.dart
│   └── storage_service.dart
│
```

```
└── state/              ← Provider/Bloc/Riverpod state managers
    ├── caption_provider.dart
    ├── auth_provider.dart
    └── tone_provider.dart
```

📦 Use `provider`, `riverpod`, or `bloc` for state management.

🧠 PYTHON BACKEND STRUCTURE (FastAPI or Flask)

```
backend/
├── main.py                 ← App entry (FastAPI or Flask instance)
├── config.py               ← Config vars, API keys, Firebase creds
│
├── routes/                 ← API endpoints
│   ├── captions.py
│   ├── auth.py
│   └── notifications.py
│
├── services/               ← Core business logic
│   ├── gpt_caption_generator.py   ← Uses OpenAI or LLM to return caption
│   ├── vision_analyzer.py      ← Uses CLIP or Google Vision
│   ├── firebase_utils.py
│   └── fcm_service.py
│
├── utils/                  ← Helper functions
```

```
│   ├── tone_mapper.py
│   └── prompt_builder.py
│
├── models/                ← Optional pydantic models / schemas
│   ├── caption.py
│   ├── user.py
│   └── response.py
│
└── requirements.txt
```

🐍 Deploy on **Render, Railway, or Vercel (Python serverless)**

💻 REACT FILE STRUCTURE (Web App)

```
src/
├── main.jsx               ← ReactDOM entry
├── App.jsx                ← Main app structure + routes
├── index.css
│
├── assets/                ← Icons, illustrations, branding
│   └── logo.svg, tone-icons.svg
│
├── components/            ← Reusable UI components
│   ├── Header.jsx
│   ├── ToneSelector.jsx
│   ├── CaptionCard.jsx
│   ├── PremiumPlan.jsx
│   └── ImageUploader.jsx
│
├── pages/                  ← Route-based pages
│   ├── LoginPage.jsx
│   ├── HomePage.jsx
│   ├── ResultsPage.jsx
│   ├── HistoryPage.jsx
│   ├── PremiumPage.jsx
│   ├── SettingsPage.jsx
│   └── AboutPage.jsx
│
├── services/               ← API fetchers
│   ├── authService.js
│   ├── captionService.js
│   ├── paymentService.js
│   └── firebase.js         ← Firebase SDK config
│
├── context/                ← React context for auth, user state
│   ├── AuthContext.jsx
│   └── AppContext.jsx
│
└── utils/
    ├── toneUtils.js
    └── toastConfig.js
```

🌐 Deploy on **Vercel or Netlify**, connected to GitHub

⏳ PROJECT TIMELINE (4–5 Weeks Solo Developer Plan)

Week	Focus Area	Tasks
Week 1	**Design + Backend Setup**	
	✅ Design Figma UI for mobile + web	
	✅ Set up Firebase (Auth, Firestore, Storage, FCM)	
	✅ Build Python backend skeleton	
	✅ Test OpenAI API + Vision tagging	
Week 2	**Build Core Flutter App (Mobile)**	
	✅ Splash, Onboarding, Login, Home Screen	
	✅ Integrate camera & image picker	
	✅ Connect to backend to get captions	

✅ Show generated captions

Week 3 Web App + Flutter Completion

✅ Build React components: upload, tones, results

✅ Complete mobile: History, Premium, Settings

✅ Add Firebase notification support (backend + Flutter)

✅ Payment integration (Razorpay / Stripe)

Week 4 Testing + Polish + Deployment

✅ Mobile: bug fixes, test edge cases

✅ Web: deploy to Vercel

✅ Backend: deploy (Railway/Render)

✅ Upload app to Play Store

✅ Build portfolio case study page

Absolutely — I'll guide you step by step through building the **entire backend for AutoText** using **Python + FastAPI + Firebase + OpenAI**. Since you're doing this solo and new to backend, I'll make it beginner-friendly, but powerful and scalable.

✅ Step-by-Step Backend Plan for AutoText

🧩 Tech Stack We'll Use

Feature	Tool / Tech
API Framework	FastAPI (lightweight, async)
Database	Firebase Firestore
Auth + Storage	Firebase Auth & Storage
Image Caption AI	OpenAI (GPT-3.5/4 via prompt)
Image Analysis	Google Cloud Vision API (or free alternative)
Deployment	Railway (easy Python hosting)
Push Notifications	Firebase Cloud Messaging (FCM)

🏁 Step 1: Backend Setup

🔧 1. Install Python & Create Virtual Environment

If you haven't already:

```
python -m venv venv

source venv/bin/activate  # On Windows: venv\Scripts\activate
```

📦 2. Install Required Libraries

```
pip install fastapi uvicorn python-multipart openai firebase-admin google-cloud-vision
python-dotenv
```

📁 3. Folder Structure (start with this)

```
backend/
├── main.py                ← App Entry Point
├── config.py              ← Environment setup & API keys
├── requirements.txt
├── .env                   ← Store your API keys safely
├── routes/
│   └── captions.py         ← /generate_caption route
├── services/
│   ├── openai_caption.py   ← Caption generator logic
│   ├── vision_service.py   ← Vision image tag extractor
│   └── firebase_service.py ← Firebase init + data functions
└── utils/
    └── prompt_builder.py   ← Create prompts using tone + tags
```

🛠️ Step 2: Initialize FastAPI App

✅ main.py

```
from fastapi import FastAPI

from routes.captions import caption_router


app = FastAPI(title="AutoText Backend")


app.include_router(caption_router, prefix="/api")


@app.get("/")

def root():

    return {"message": "AutoText backend is running"}
```

🔐 Step 3: Configure Environment

✅ .env

```
OPENAI_API_KEY=sk-xxxxxxxxxxxx

GOOGLE_VISION_API_KEY=your_google_api_key

FIREBASE_JSON_PATH=your_service_account.json
```

✅ config.py

```
import os

from dotenv import load_dotenv

import openai


load_dotenv()
```

```
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

VISION_API_KEY = os.getenv("GOOGLE_VISION_API_KEY")

FIREBASE_JSON_PATH = os.getenv("FIREBASE_JSON_PATH")


openai.api_key = OPENAI_API_KEY
```

🔍 Step 4: Image Analysis using Vision API

✅ `services/vision_service.py`

```
from google.cloud import vision

import io


def extract_image_labels(image_bytes):

    client = vision.ImageAnnotatorClient()

    image = vision.Image(content=image_bytes)

    response = client.label_detection(image=image)

    labels = [label.description for label in response.label_annotations]

    return labels
```

✍️ Step 5: AI Caption Generator (OpenAI)

✅ `services/openai_caption.py`

```
import openai

from config import OPENAI_API_KEY
```

```
def generate_caption(prompt: str):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "user", "content": prompt}
        ],
        temperature=0.8
    )
    return response.choices[0].message.content.strip()
```

💬 Step 6: Prompt Builder

✅ `utils/prompt_builder.py`

```
def build_caption_prompt(tone, tags):
    tag_text = ", ".join(tags)
    return f"Write a {tone} social media caption for a photo that includes {tag_text}. Keep it short, fun, and engaging."
```

🔁 Step 7: API Route to Generate Captions

✅ `routes/captions.py`

```
from fastapi import APIRouter, UploadFile, File, Form
from services.vision_service import extract_image_labels
from services.openai_caption import generate_caption
from utils.prompt_builder import build_caption_prompt


caption_router = APIRouter()


@caption_router.post("/generate_caption")
async def generate_caption_route(file: UploadFile = File(...), tone: str = Form(...)):
    image_bytes = await file.read()
    tags = extract_image_labels(image_bytes)
    prompt = build_caption_prompt(tone, tags)
    caption = generate_caption(prompt)
    return {"tags": tags, "caption": caption}
```

Test this using Postman or frontend.

🔥 Step 8: Add Firebase Support

✅ `services/firebase_service.py`

```
import firebase_admin
from firebase_admin import credentials, firestore, storage
from config import FIREBASE_JSON_PATH


cred = credentials.Certificate(FIREBASE_JSON_PATH)
firebase_admin.initialize_app(cred, {'storageBucket': '<your_bucket>.appspot.com'})


db = firestore.client()
bucket = storage.bucket()
```

```
def save_caption(user_id, caption_data):
    ref = db.collection('Users').document(user_id).collection('captions')
    ref.add(caption_data)
```

You can now:

- Store generated captions
- Upload image to Firebase Storage
- Link the URL to Firestore

✅ With this setup, your **backend is fully functional** to:

- Accept images
- Analyze with Vision API
- Build a prompt
- Call OpenAI to generate a caption
- Return caption
- Store it in Firebase

🚀 NEXT STEPS (Tell me what you want next):

1. Add **Firebase Auth (Token-based user login)**
2. Upload image to **Firebase Storage**
3. Return image URL with caption
4. Build **/get_history** endpoint
5. Add **notifications**

6. Add **payment validation API**
7. Deployment on **Railway**

Just tell me where to go from here and I'll help you write that step.

We're building this backend cleanly, one layer at a time.