📱 🧠 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:

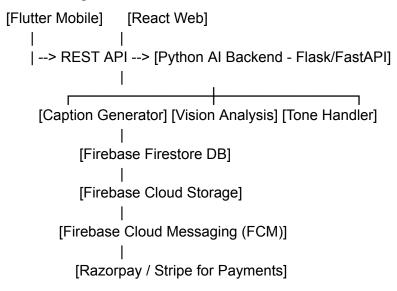
People often struggle to write creative, mood-matching captions for their social media posts.

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



→ 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	V	X
Login/Signup	V	V
Home (Upload + Tone)	V	V
Captions Results	V	V
History	V	V
Premium Plans	V	V
Profile / Settings	V	V
Admin Panel	X	V
About Us	V	V

5. Icons, Images, & UI Design Guide

- Name of the second secon
- Images: Custom illustrations (for onboarding), user-uploaded photos
- | Icons: Feather Icons, FontAwesome, or Material Icons

• ***** Tone Icons:

- 😂 Funny → 😆
- **Property** Romantic → **Property**
- Poetic →
- Sarcastic → ••

⊕ 6. Notifications

Туре	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 ເ≥"
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. Q Image Analysis (Vision AI):

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

b. F Caption Generator (Text AI):

OpenAl 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
 - o 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/

☐ ppt_generator.py, vision_analyzer.py

/utils/

L 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

o 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):
 - o 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 \rightarrow Home \rightarrow Generate \rightarrow 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.
- - 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. Rirebase Database Structure
- 2. Tirestore Security Rules
- 3. Sigma Layout Tips (for each screen)
- 4. 🔬 Al Prompt Templates (tone-based captioning)
- 5. H Local Data Caching Strategy
- 6. Deployment Plan (Mobile, Web, Backend)
- 7. Future Features & Scalability Plan

2 1. Firebase Firestore Database Structure

Structure:

Users (Collection)

└ userId (Document)

hame: 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 == userld;
  }
}
}
```

🎨 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. Al 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
Romanti c	"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. V Build and deploy **Python Backend** (FastAPI) on:
 - o Railway / Render / DigitalOcean / Vercel Serverless Functions
 - Use a .env for GPT/Google Vision keys
- 2. Set up **Firebase** project:
 - Firestore
 - Storage
 - FCM
 - o Auth
- 3. Integrate backend API in Flutter app
- 4. Deploy Flutter app to Play Store
- 5. We 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

Why It's Useful **Feature**

Hashtag Al 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

Marian Pilatter Dev Camera, uploads, design

React Dev
Web frontend

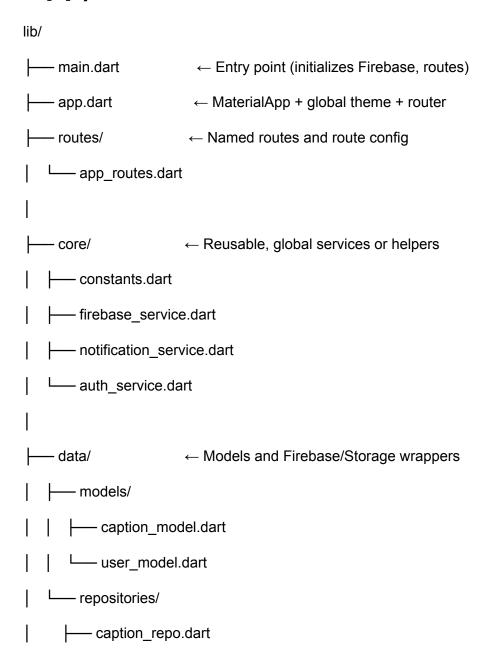
Growth Marketer Push notifications, tone

ideas

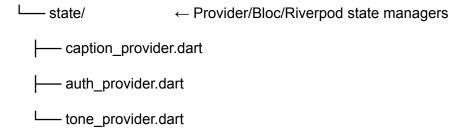
Perfect. Here's your complete **file/folder structure** for all 3 parts of the AutoText project — Flutter app, Python backend, and React web — with **detailed folder roles, filenames, and responsibilities**.

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)



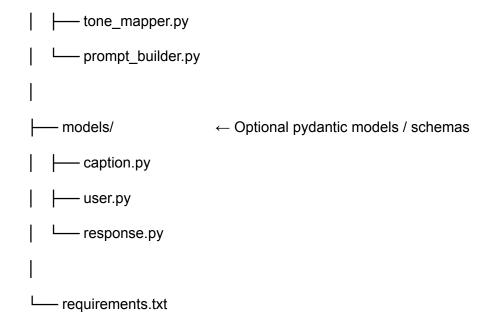
```
--- auth_repo.dart
  history_repo.dart
— ui/
              ← UI layer (screens + widgets)
— onboarding_screen.dart
  login_screen.dart
— home_screen.dart
 ---- caption_result_screen.dart
  — history_screen.dart
 premium_screen.dart
  ---- settings_screen.dart
  L— about_screen.dart
widgets/
  — tone_selector.dart
  — 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
```



Use provider, riverpod, or bloc for state management.

PYTHON BACKEND STRUCTURE (FastAPI or Flask)

ba	ckend/	
-	— main.py	← App entry (FastAPI or Flask instance)
\vdash	— config.py	← Config vars, API keys, Firebase creds
-	— routes/	← API endpoints
	— captions.py	
	— auth.py	
	└── notifications.py	
\vdash	— services/	← Core business logic
	— gpt_caption_gene	rator.py ← Uses OpenAl or LLM to return caption
	vision_analyzer.py	← Uses CLIP or Google Vision
	firebase_utils.py	
	L fcm_service.py	
_	— utils/	← Helper functions



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	
I	
assets/	$\leftarrow \text{Icons, illustrations, branding}$
logo.svg, tone-ic	cons.svg
I	
components/	← Reusable UI components
│	
ToneSelector.jsx	ζ.
CaptionCard.jsx	

	—— PremiumPlan.jsx		
	└── ImageUploader.jsx		
H	— pages/	← Route-based pages	
	LoginPage.jsx		
	├── HomePage.jsx		
	├── HistoryPage.jsx		
	PremiumPage.js:	x	
	SettingsPage.jsx		
	L— AboutPage.jsx		
\vdash	— services/	← API fetchers	
	— authService.js		
	captionService.js	3	
	paymentService.	js	
	L—firebase.js	← Firebase SDK config	
H	— context/	← React context for auth, user state	
	— AuthContext.jsx		
	L—AppContext.jsx		
L	— utils/		
	toneUtils.js		
	└── toastConfig.js		

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

Week **Focus Area** Task S Week 1 Design + Backend Setup Design Figma UI for mobile + web Set up Firebase (Auth, Firestore, Storage, FCM) ✓ Build Python backend skeleton Test OpenAl 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 Al OpenAl (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)



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/	
	n.py ← Caption generator logic
	py ← Vision image tag extractor
	ce.py ← Firebase init + data functions
L— utils/	
└── prompt_builder.	py ← Create prompts using tone + tags

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"}



Rep 3: Configure Environment



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: Al Caption Generator (OpenAl)

services/openai_caption.py

import openai

from config import OPENAI_API_KEY

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 OpenAl 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.