
🔐 Step 1: Firebase User Authentication in Backend (Protect APIs)

✅ Goal:

Ensure **only authenticated users** can call your APIs (e.g. `/generate_caption`, `/get_history`, etc.) using their **Firebase ID token**.

⚙️ How it Works

1. Your Flutter/React frontend uses Firebase Auth to log in (Google/Email/Phone).
2. It gets a secure **ID Token** from Firebase.
3. Every API request includes this ID token in the **`Authorization` header**.
4. Backend verifies the token → gets the `uid`.

✅ Backend Implementation (FastAPI + Firebase)

🔧 1. Update `firebase_service.py`

Add this function:

```
from firebase_admin import auth
from fastapi import HTTPException, Header

def verify_firebase_token(auth_header: str):
    if not auth_header:
        raise HTTPException(status_code=401, detail="Authorization header missing")

    try:
        id_token = auth_header.split("Bearer ")[1]
        decoded_token = auth.verify_id_token(id_token)
        return decoded_token['uid']
    except Exception as e:
        raise HTTPException(status_code=401, detail="Invalid Firebase ID token")
```

🔒 2. Use in `generate_caption` route

Update `routes/captions.py`:

```
from fastapi import APIRouter, UploadFile, File, Form, Header
from services.firebase_service import verify_firebase_token
# (Other imports remain)

@caption_router.post("/generate_caption")
async def generate_caption_route(
    file: UploadFile = File(...),
    tone: str = Form(...),
    authorization: str = Header(None)
):
    # 🔐 Check auth
    user_id = verify_firebase_token(authorization)

    # 🧠 Image tags → prompt → caption
    image_bytes = await file.read()
    tags = extract_image_labels(image_bytes)
    prompt = build_caption_prompt(tone, tags)
    caption = generate_caption(prompt)

    # Optionally save to Firestore
    return {
        "userId": user_id,
        "tags": tags,
        "caption": caption
    }
```

🚀 Frontend Side (Flutter / React)

When calling this API:

- Make sure you're logged in using Firebase Auth.
- Use `getIdToken()` and attach it in headers:

🔐 Flutter Example:

```
final user = FirebaseAuth.instance.currentUser;
final token = await user?.getIdToken();

final response = await http.post(
  Uri.parse("https://your-backend/api/generate_caption"),
  headers: {
    'Authorization': 'Bearer $token',
  },
  body: {
    'tone': 'funny',
    'file': yourImageFile,
  },
);
```

✅ Done!

You now have:

- 🔒 Secure API access
- 👤 User identified by `uid`
- 🧱 Foundation for saving captions per user

🧪 Test Checklist:

- ✅ Try with a valid token → should work
- ❌ Try with no token → get 401
- ❌ Try with fake token → get 401

☁️ Step 2: Uploading Images to Firebase Storage (from Python Backend)

🎯 Goal:

When the user uploads a photo:

- We upload it to **Firebase Storage**
- Get a public image **URL**
- Store it later in Firestore with the caption

🔧 Pre-setup: Firebase Storage Rules

Make sure your storage allows authenticated access.

Go to **Firebase Console** → **Storage** → **Rules** and set:

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

🛠️ Update Backend to Upload Image

✅ `firebase_service.py`

Add this image upload function:

```
import time
from firebase_admin import storage
```

```
def upload_image_to_storage(user_id: str, image_bytes: bytes, filename: str) -> str:
    bucket = storage.bucket()
    timestamp = int(time.time())
    blob_path = f"captions/{user_id}/{timestamp}_{filename}"
    blob = bucket.blob(blob_path)

    blob.upload_from_string(image_bytes, content_type='image/jpeg')

    # Make public URL (or use token-based access if preferred)
    blob.make_public()
    return blob.public_url
```

> 🔐 Optional: You can use `blob.generate_signed_url()` if you prefer **private URLs**.

🧩 Now Update `/generate_caption` API

In `routes/captions.py`, use the new upload function:

```
from services.firebase_service import upload_image_to_storage


@caption_router.post("/generate_caption")
async def generate_caption_route(
    file: UploadFile = File(...),
    tone: str = Form(...),
    authorization: str = Header(None)
):
    user_id = verify_firebase_token(authorization)
```

```
image_bytes = await file.read()


# ⬆ Upload image
image_url = upload_image_to_storage(user_id, image_bytes, file.filename)


# 🔍 Analyze → prompt → generate caption
tags = extract_image_labels(image_bytes)
prompt = build_caption_prompt(tone, tags)
caption = generate_caption(prompt)


return {
    "userId": user_id,
    "imageUrl": image_url,
    "tags": tags,
    "caption": caption
}
```

🧪 Test Case:

1. Upload an image via frontend.
2. Backend stores it in `captions/userId/` folder.
3. Public URL returned in the response.

✅ This URL can be shown in the frontend and stored in Firestore with the caption.

✅ Output Sample:

```
{
  "userId": "Xyz123",
  "imageUrl": "https://firebasestorage.googleapis.com/...",
  "tags": ["beach", "sunset", "couple"],
  "caption": "Together is a wonderful place to be 🌅"
}
```

🔥 Your backend can now:

- Authenticate users
- Accept and upload images to cloud
- Generate an AI caption
- Return everything for the frontend

🧾 Step 3: Saving & Fetching Caption History (Firestore)

🎯 Goal:

1. When a caption is generated, we save it in **Firestore** under the user's document.
2. We create a new route to fetch all **past captions** (history).

📂 Firestore Structure Recap

```
Users (Collection)
 └─ {userId} (Document)
     └─ captions (Subcollection)
         └─ {captionId} (Document)
             ├─ imageUrl
             ├─ caption
             ├─ tone
             ├─ tags
             └─ timestamp
```

✅ Step A: Save Caption to Firestore

🔧 Add to `firebase_service.py`:

```
from datetime import datetime


def save_caption_to_firestore(user_id: str, caption: str, image_url: str, tags: list, tone: str):
    ref = db.collection('Users').document(user_id).collection('captions')
    ref.add({
        'caption': caption,
        'imageUrl': image_url,
        'tone': tone,
        'tags': tags,
        'timestamp': datetime.utcnow()
    })
```

🔁 Update `/generate_caption` route to store it

In `routes/captions.py`:

```
from services.firebase_service import save_caption_to_firestore


# Inside the route after generating the caption:
save_caption_to_firestore(user_id, caption, image_url, tags, tone)
```

✅ Step B: Get Caption History API

📁 New Route: `/get_history`

In `routes/captions.py`, add this:

```
@caption_router.get("/get_history")
def get_caption_history(authorization: str = Header(None)):
    user_id = verify_firebase_token(authorization)

    ref = db.collection('Users').document(user_id).collection('captions')
    docs = ref.order_by('timestamp', direction=firestore.Query.DESCENDING).stream()

    history = []
    for doc in docs:
        data = doc.to_dict()
        data['id'] = doc.id
        history.append(data)

    return {"history": history}
```

🧪 Output Sample:

```
{
  "history": [
    {
      "caption": "Living our best life 🌴",
      "imageUrl": "https://...jpg",
      "tone": "funny",
      "tags": ["beach", "friends"],
      "timestamp": "2024-06-25T17:03:19.015Z"
    },
    ...
  ]
}
```

✅ Test Plan:

1. ✅ Generate a caption → should save it to Firestore.
2. ✅ Call `/get_history` → get all past captions (sorted by newest).
3. ✅ Use this to show caption cards on the **History screen** in mobile/web.

✅ Now you've completed:

- 🔐 Auth

- 🌥 Image upload
- 🧾 Save & fetch caption history

📲 Step 4: Push Notifications (Using FCM + Custom Reminders)

🎯 Goal:

Send **fun, Hindi-style notifications** if the user hasn't opened the app in **3/6/9/15/30 days**, like:

- 🗓️ "Bhul toh nahi gaye hume? 😢" (3 days)
- "6 din ho gaye, caption to dekh le baba!"
- "Kya hum yaad nahi aaye?" (9 days)
- "15 din baad mulakat 😮"
- "30 din ke vanvaas ke baad bhi wapas aa gaye?" 😂

We'll also send **promo notifications** like:

- "Unlock unlimited captions — get AutoText Premium ✨"
- "New tone added: Savage 🔥 Try now!"

✅ Step A: Setup Firebase Cloud Messaging (FCM)

🔧 In Firebase Console:

1. Go to **Project Settings > Cloud Messaging**
2. Copy your **Server key**

3. Add it to `.env`:

```
FCM_SERVER_KEY=AAAA...your_key
```

📦 Step B: Store User Device Tokens

🔁 Frontend (Flutter/React):

When user logs in:

```
FirebaseMessaging messaging = FirebaseMessaging.instance;

String? token = await messaging.getToken();


// Send token to backend with /save_token
```

🔧 Backend route: `routes/notifications.py`

```
from fastapi import APIRouter, Header

from services.firebase_service import verify_firebase_token, db


notify_router = APIRouter()


@notify_router.post("/save_token")

def save_fcm_token(token: str, authorization: str = Header(None)):

    user_id = verify_firebase_token(authorization)

    user_ref = db.collection("Users").document(user_id)

    user_ref.set({"deviceToken": token}, merge=True)

    return {"message": "Token saved"}
```

🚀 Step C: Send Notification Function

✅ In `services/fcm_service.py`:

```
import requests

import os

from config import FCM_SERVER_KEY


def send_notification(token, title, body):

    headers = {

        'Content-Type': 'application/json',

        'Authorization': f'key={FCM_SERVER_KEY}'

    }


    payload = {

        "to": token,

        "notification": {

            "title": title,

            "body": body

        }

    }


    response = requests.post("https://fcm.googleapis.com/fcm/send", json=payload,
headers=headers)

    return response.json()
```

🧠 Step D: Scheduled Notification Logic (Run Daily)

You can run this manually or via cron/Cloud Function.

✅ `scripts/send_reminders.py` (Example Script):

```
from firebase_admin import firestore

from datetime import datetime, timedelta

from services.fcm_service import send_notification

from firebase_service import db


reminder_days = {

    3: "Bhul toh nahi gaye hume? 😢",

    6: "6 din ho gaye, caption to dekh le baba!",

    9: "Kya hum yaad nahi aaye?",

    15: "15 din baad mulakat 😮",

    30: "30 din ke vanvaas ke baad bhi wapas aa gaye? 😂"

}


def run_reminder_job():

    users = db.collection("Users").stream()

    for user in users:

        data = user.to_dict()

        last_active = data.get("lastActive")

        token = data.get("deviceToken")


        if not last_active or not token:

            continue
        days_inactive = (datetime.utcnow() - last_active.replace(tzinfo=None)).days

        if days_inactive in reminder_days:
            message = reminder_days[days_inactive]
            send_notification(token, "AutoText Missing You 💌", message)

if __name__ == "__main__":
    run_reminder_job()
```

🔁 Optional: Update lastActive in every API

In your `generate_caption` route or any activity:

```
db.collection("Users").document(user_id).update({
    "lastActive": datetime.utcnow()
})
```

✅ Result:

- Users get fun reminders automatically at inactivity intervals
- You can also trigger premium offers or new feature alerts
- Feels personal and builds retention

📦 ✅ Done! Your backend can now:

- Save & manage FCM tokens

- Send scheduled & manual notifications

💰 Step 5: Premium Access Validation (Stripe or Razorpay)

🎯 Goal:

Let users buy premium → verify the payment on backend → mark them as `isPremium: true` in Firebase → unlock unlimited captions.

You'll show/hide premium features based on this flag.

✅ Option A: Razorpay (for India 🇮🇳)

🔧 Step A1: Setup Razorpay

1. Go to https://razorpay.com → Create account
2. Get your:
    - **API Key ID**
    - **API Secret**
3. Add to `.env`:

RAZORPAY_KEY=rzp_test_abc123

RAZORPAY_SECRET=xyz_secret_key

✅ Step A2: Python Backend Validation

📦 Install:

```
pip install razorpay
```

🔧 **Add to `config.py`:**

```
import razorpay


RAZORPAY_CLIENT = razorpay.Client(auth=(os.getenv("RAZORPAY_KEY"),
os.getenv("RAZORPAY_SECRET")))
```

✅ Step A3: API to verify Razorpay payment

Add in `routes/payments.py`:

```
from fastapi import APIRouter, Form, Header

from services.firebase_service import verify_firebase_token, db

from config import RAZORPAY_CLIENT


payment_router = APIRouter()


@payment_router.post("/verify_payment")

def verify_payment_route(

    razorpay_payment_id: str = Form(...),

    razorpay_order_id: str = Form(...),

    razorpay_signature: str = Form(...),

    authorization: str = Header(None)

):

    user_id = verify_firebase_token(authorization)
```

```
params_dict = {
    'razorpay_payment_id': razorpay_payment_id,
    'razorpay_order_id': razorpay_order_id,
    'razorpay_signature': razorpay_signature
}

try:
    RAZORPAY_CLIENT.utility.verify_payment_signature(params_dict)
except:
    raise HTTPException(status_code=400, detail="Payment verification failed")

# ✅ Update Firestore
db.collection("Users").document(user_id).set({
    "isPremium": True
}, merge=True)

return {"message": "Premium activated!"}
```

🔁 Frontend Workflow:

1. Call Razorpay SDK from Flutter or React
2. After successful payment, Razorpay sends:
    - `payment_id`
    - `order_id`
    - `signature`

3. Send these 3 values + ID token to `/verify_payment`

✅ Option B: Stripe (International)

If you're targeting global users, Stripe is easier and safer.

Let me know if you want the full Stripe setup (webhook + client-side checkout) — I'll guide that too.

✅ Checking Premium Access in Backend

Anywhere in your backend (e.g. `/generate_caption`), you can check:

```
user_ref = db.collection("Users").document(user_id).get()
if not user_ref.exists:
    raise HTTPException(status_code=404, detail="User not found")

user_data = user_ref.to_dict()
is_premium = user_data.get("isPremium", False)

# Optional: limit free users to 3/day
if not is_premium:
    captions_today = db.collection("Users").document(user_id).collection("captions")\
        .where("timestamp", ">", datetime.utcnow().replace(hour=0, minute=0, second=0))\
        .stream()
    if len(list(captions_today)) >= 3:
        raise HTTPException(status_code=403, detail="Free limit exceeded. Buy Premium.")
```

✅ Summary: Your Full Backend Now Has

Feature	Status
Firebase Auth Token	✅ Done
Upload Image to Cloud	✅ Done
AI Caption Generation	✅ Done
Store Caption History	✅ Done
FCM Notifications	✅ Done
Premium Validation	✅ Done
