Commands and changes to get the credit management right

#1 Update Supabase with that SQL script:

```
alter table public.users
  add column if not exists free_grant_remaining integer not null default 3,
  add column if not exists subscription_period_start timestamptz,
  add column if not exists subscription_period_end   timestamptz;

alter table public.images
  add column if not exists is_initial boolean not null default false;
```

#2 Delete these two helper functions

```
monthly_image_usage
plan_limit
```

#3 Add this new helper function

```
def images_count_since(user_id: str, since_iso: str | None) -> int:
    """
    Count billable images for a user since a given ISO timestamp.
    Excludes the initial influencer image (is_initial = True).
    """
    q = supabase.table("images").select("id", count="exact") \
        .eq("user_id", user_id) \
        .eq("is_initial", False)
    if since_iso:
        q = q.gte("created_at", since_iso)
    r = q.execute()
    return int(r.count or 0)
```

#4 api/me (replace whole function)

```
@app.get("/api/me")
def me():
    user = get_user_from_auth_header()
    if not user:
        return jsonify({"error": "unauthorized"}), 401

    ensure_user_row_if_missing(user.id, getattr(user, "email", None))
    row = supabase.table("users").select("*").eq("id", user.id).single().execute().data or {}

    plan = row.get("plan", "free")
    if plan == "pro":
        # Stripe period–based credits. If period not yet stored, show full 20.
        period_start = row.get("subscription_period_start")
        if period_start:
            used = images_count_since(user.id, period_start)
        else:
            used = 0
        credits = max(20 - used, 0)
    else:
        # Free = one-time bucket stored on the user row (default 3)
        credits = max(int(row.get("free_grant_remaining", 3)), 0)

    return jsonify({"plan": plan, "credits": credits})
```

#5 stripe webhook (replace whole function)

```
@app.post("/api/stripe/webhook")
def stripe_webhook():
    payload = request.data
    sig = request.headers.get("Stripe-Signature")
    try:
        event = stripe.Webhook.construct_event(payload, sig, STRIPE_WEBHOOK_SECRET)
    except Exception as e:
        return jsonify({"error": str(e)}), 400

    et = event["type"]
    obj = event["data"]["object"]

    def set_plan_by_status(customer_id: str, stripe_status: str, sub_obj=None):
        internal_status = "active" if stripe_status in ("active", "trialing") \
            else ("past_due" if stripe_status == "past_due" else "canceled")
        plan = "pro" if internal_status == "active" else "free"

        res = supabase.table("users").select("id").eq("stripe_customer_id",
customer_id).single().execute()
        if not res.data:
            return
        uid = res.data["id"]

        update = {
            "plan": plan,
            "subscription_status": internal_status
        }

        # When we have a subscription object, capture the billing period window.
        if sub_obj and sub_obj.get("current_period_start") and sub_obj.get("current_period_end"):
            from_ts = datetime.utcfromtimestamp(sub_obj["current_period_start"]).isoformat()
            to_ts   = datetime.utcfromtimestamp(sub_obj["current_period_end"]).isoformat()
            update["subscription_period_start"] = from_ts if plan == "pro" else None
            update["subscription_period_end"]   = to_ts if plan == "pro" else None
        elif plan != "pro":
            # Clear period fields when not active
            update["subscription_period_start"] = None
            update["subscription_period_end"] = None

        supabase.table("users").update(update).eq("id", uid).execute()

    if et == "checkout.session.completed":
        uid = (obj.get("metadata") or {}).get("supabase_uid")
        if uid and obj.get("customer"):
            supabase.table("users").update({"stripe_customer_id": obj["customer"]}).eq("id",
uid).execute()

    elif et in ("customer.subscription.created", "customer.subscription.updated"):
        set_plan_by_status(obj["customer"], obj.get("status", ""), obj)

    elif et == "customer.subscription.deleted":
        set_plan_by_status(obj["customer"], "canceled", obj)

    return jsonify({"received": True})
```

#6 finalize_influencer (replace whole function)

```
@app.post("/api/influencer/finalize")
def finalize_influencer():
    # n8n → Flask callback
    secret = request.headers.get("X-Callback-Secret", "") or (request.json or
{}).get("callback_secret", "")
    if secret != N8N_CALLBACK_SECRET:
        return jsonify({"error": "unauthorized"}), 401

    data = request.get_json(silent=True) or {}
    influencer_id = data.get("influencer_id")
    base_prompt = data.get("base_prompt")
    seed = data.get("seed")
    replicate_image_url = data.get("image_url")

    if not influencer_id or not base_prompt or seed is None or not replicate_image_url:
        return jsonify({"error": "missing_fields"}), 400

    # Get owner
    row = supabase.table("influencers").select("user_id").eq("id",
influencer_id).single().execute().data
    if not row:
        return jsonify({"error": "unknown_influencer"}), 404
    user_id = row["user_id"]

    # Save image to /media/<user_id>/...
    try:
        _, public_url = save_image_to_media(replicate_image_url, user_id)
    except Exception as e:
        return jsonify({"error": f"image_download_failed: {e}"}), 500

    now_iso = datetime.utcnow().isoformat()

    # Update influencer (lock + store assets)
    supabase.table("influencers").update({
        "base_prompt": base_prompt,
        "seed": seed,
        "initial_image_url": public_url,
        "is_locked": True,
        "created_at": now_iso
    }).eq("id", influencer_id).execute()

    # Record the initial image for history/preview, but mark as non-billable
    supabase.table("images").insert({
        "user_id": user_id,
        "influencer_id": influencer_id,
        "prompt_final": base_prompt,
        "url": public_url,
        "created_at": now_iso,
        "is_initial": True
    }).execute()

    return jsonify({"ok": True})
```

#2 Alternatively just switch out the file I provide:
app.py

#3 restart the app:
sudo systemctl daemon-reload
sudo systemctl restart gptsweetheart
sudo systemctl status gptsweetheart