Initial Prompt I used to start the SAAS development with ChatGPT5

My goal is to build a simple but functional SAAS with n8n, a cloud server, Flask and Lovable (Frontend Development).

I already set up a Vultr cloud server and installed n8n on it.

I show you now the general Idea of the SAAS I want to build:

The idea is that a user can sign in for free and set up an account. After the email got confirmed he is automatically redirected to the login. When he logs in, he is in his dashboard and automatically on a free account. That allows him to create his AI Girl (select some parameters and a prompt gets created in the backend) and then an Image of that girl appears in a certain area of the dashboard. That image can't be reversed, its a one time process after setting up the account. With the free account he can create three images per month and endless chatting with the girl (image creation and chatting is done via n8n workflows). When the user subscribes to a pro subscription he gets 20 image creations per month. Every month the credits get set back to 20 until the subscription gest cancelled. An entire subscription management will be implemented as well, via Stripe. (Already have experience with all that as I already built my own SAAS the exact same way). In the dashboard the user sees in a gallery all the images he created.

Here you see the general structure of the SAAS:

1. User Onboarding Supabase sign-up/login Stripe subscription for free and pro account
2. AI Influencer Creation User picks a name, short bio, and vibe/style Backend calls n8n workflow that calls Replicate → creates first image → stores: Seed number Base prompt Generated image URL
3. Chat with Influencer Chat interface (frontend Lovable) Messages go to backend and another n8n workflow → OpenAI API (with influencer persona prompt) Responses stored in DB
4. Generate More Images User requests e.g., "in a red dress on the beach" Backend sends base prompt + seed + user modification to Replicate New image shown in chat or image gallery
5. Usage Limits Free tier = unlimited messages / 1 initial image plus 3 extra images Pro tier = unlimited messages / 20 images per Month Quota tracking in Supabase

Components involved and system Design:

-Flask app running on Vultr Server
-n8n running on Vultr Server
-Frontend running on Vultr Server (Frontend will be designed with Lovable and then connected to the backend)

Now that you saw the entire Game Plan, I need you to do the following: Tell me the simplest but also truly working way to build my SAAS.

I actually don't see any challenges with exception of the chatting part. Tell me how I can make sure, that each user gets identified correctly and can continue with the same chat he left earlier.

I guess we need a job queue in the Flask app to generally process different requests from different users but I don't see yet, how the chats can be continued by basically using one n8n workflow in the background.

If possible I want to do the entire chatting and image creation with n8n (and replicate) to also give n8n enough exposure in the course. Tell me how I best handle that chatting part.

Take as much time as you. need to truly think this through and understand what I actually need. Its very important.

That's the order to build (and document) the entire SAAS building process recommended by ChatGPT

1. **Auth + Plans**
- Supabase auth (email link), store plan in users.
- Stripe webhook flips plan between free/pro.

2. **Create Influencer**
- Post name/bio/vibe → Flask → n8n → Replicate → Supabase update → dashboard shows image.
- Lock after creation.

3. **Chat**
- Show the one-chat-per-user idea.
- Send a message, see it saved, n8n fetches history, returns reply, saved again.

4. **Generate More Images**
- Show quota logic (computed per month).
- Free: 4 total; Pro: 20. Upgrade in Stripe and show the difference immediately.

5. **Security**
- Show HMAC signatures on all Flask→n8n calls.
- Show RLS policies for end-user safety.
- Show n8n storing only the service role key, never exposed.

After some follow up Questions I instructed ChatGPT like that:

„Alright I got all that. Lets set up supabase and stripe. Tell me if you need to see the initial idea again, so you know which tables we need at supabase"

Thats the SQL script I got back from ChatGPT to set up the PostgrSGL data base at Supabase (Skript also available in my Github Repo)

```
-- Enable needed extensions
create extension if not exists "uuid-ossp";
create extension if not exists pgcrypto;  -- for gen_random_uuid()

-- USERS: mirrors auth.users
create table if not exists public.users (
  id uuid primary key references auth.users(id) on delete cascade,
  plan text not null default 'free',              -- 'free' | 'pro'
  subscription_status text,                       -- 'active' | 'canceled' | 'past_due' | null
  stripe_customer_id text,
  created_at timestamptz not null default now()
);

alter table public.users enable row level security;

create policy "read own users row"
on public.users for select
to authenticated
using (id = auth.uid());

create policy "update own users row (limited)"
on public.users for update
to authenticated
using (id = auth.uid())
with check (id = auth.uid());
-- Note: Your backend/n8n will use the SERVICE ROLE key and bypass RLS anyway.

-- INFLUENCER: one per user (locked after creation)
create table if not exists public.influencers (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references public.users(id) on delete cascade,
  name text not null,
  bio text not null,
  vibe text not null,
  base_prompt text,
  seed text,
  initial_image_url text,
  is_locked boolean not null default false,
  created_at timestamptz not null default now(),
  unique(user_id)  -- enforce one influencer per user
);

alter table public.influencers enable row level security;

create policy "select own influencer"
on public.influencers for select
to authenticated
using (user_id = auth.uid());

create policy "insert own influencer"
on public.influencers for insert
to authenticated
with check (user_id = auth.uid());
```

```
create policy "update own influencer until locked"
on public.influencers for update
to authenticated
using (user_id = auth.uid() and is_locked = false)
with check (user_id = auth.uid());

-- CHATS: exactly one chat tied to the influencer
create table if not exists public.chats (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references public.users(id) on delete cascade,
  influencer_id uuid not null references public.influencers(id) on delete cascade,
  created_at timestamptz not null default now(),
  unique(user_id, influencer_id)
);

alter table public.chats enable row level security;

create policy "select own chats"
on public.chats for select
to authenticated
using (user_id = auth.uid());

create policy "insert own chat"
on public.chats for insert
to authenticated
with check (user_id = auth.uid());

-- MESSAGES
create table if not exists public.messages (
  id uuid primary key default gen_random_uuid(),
  chat_id uuid not null references public.chats(id) on delete cascade,
  role text not null check (role in ('user','assistant','system')),
  content text not null,
  created_at timestamptz not null default now()
);

alter table public.messages enable row level security;

-- Users can read messages only for their chats
create policy "select messages for own chats"
on public.messages for select
to authenticated
using (exists(
  select 1 from public.chats c
  where c.id = messages.chat_id and c.user_id = auth.uid()
));

-- Users can insert messages only into their chats
create policy "insert messages into own chats"
on public.messages for insert
to authenticated
with check (exists(
  select 1 from public.chats c
  where c.id = messages.chat_id and c.user_id = auth.uid()
));
```

```
-- IMAGES
create table if not exists public.images (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references public.users(id) on delete cascade,
  influencer_id uuid not null references public.influencers(id) on delete cascade,
  prompt_final text,
  url text not null,
  created_at timestamptz not null default now()
);

alter table public.images enable row level security;

create policy "select own images"
on public.images for select
to authenticated
using (user_id = auth.uid());

create policy "insert own images"
on public.images for insert
to authenticated
with check (user_id = auth.uid());

-- OPTIONAL: fast monthly usage view + RPC
create or replace view public.images_this_month as
select
  user_id,
  count(*)::int as count
from public.images
where created_at >= date_trunc('month', now())
group by user_id;

create or replace function public.monthly_image_count(u uuid)
returns int language sql stable as $$
  select coalesce(count(*),0)::int
  from public.images
  where user_id = u
    and created_at >= date_trunc('month', now());
$$;
```