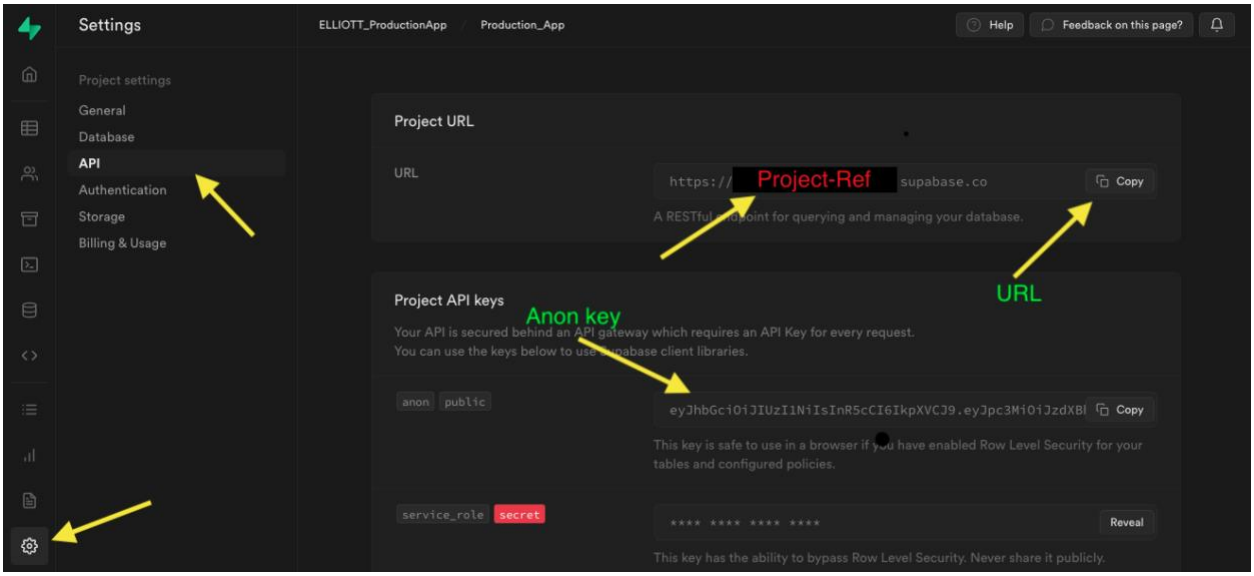
HAPPY DAYS README GUIDE

ACCOUNTS REQUIREMENTS

- Supabase free account
- Cloudflare workers
- GitHub

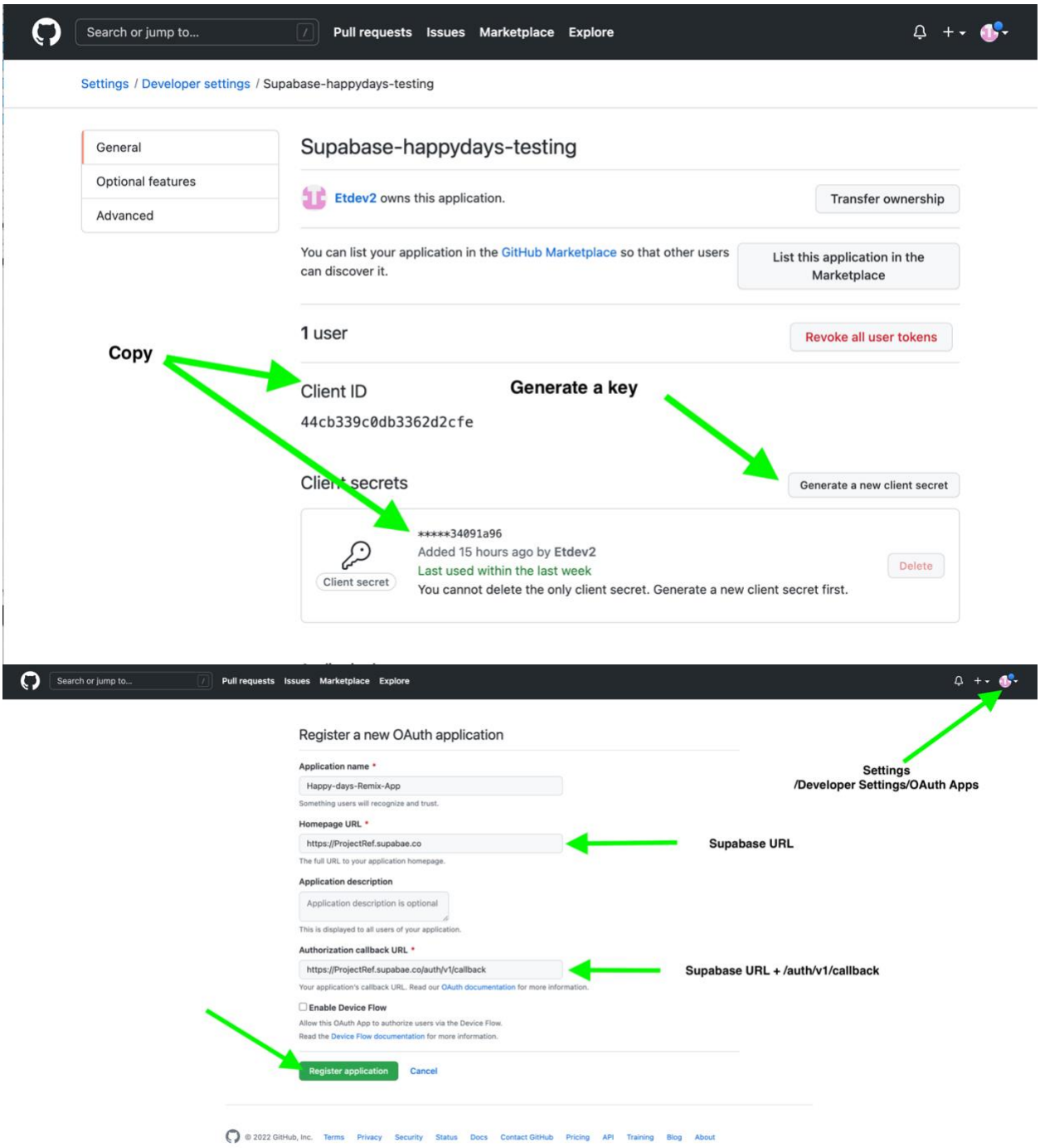
SUPABASE REFERENCES (Copy these to note pad will be used many times in the guide) (See screenshots below as a visual reference to locate)

- Supabase URL (supabase/settings/api)
- Supabase Anon Key (Supabase/setting/api)
- Supabase Project Ref (supabase/settings/api)



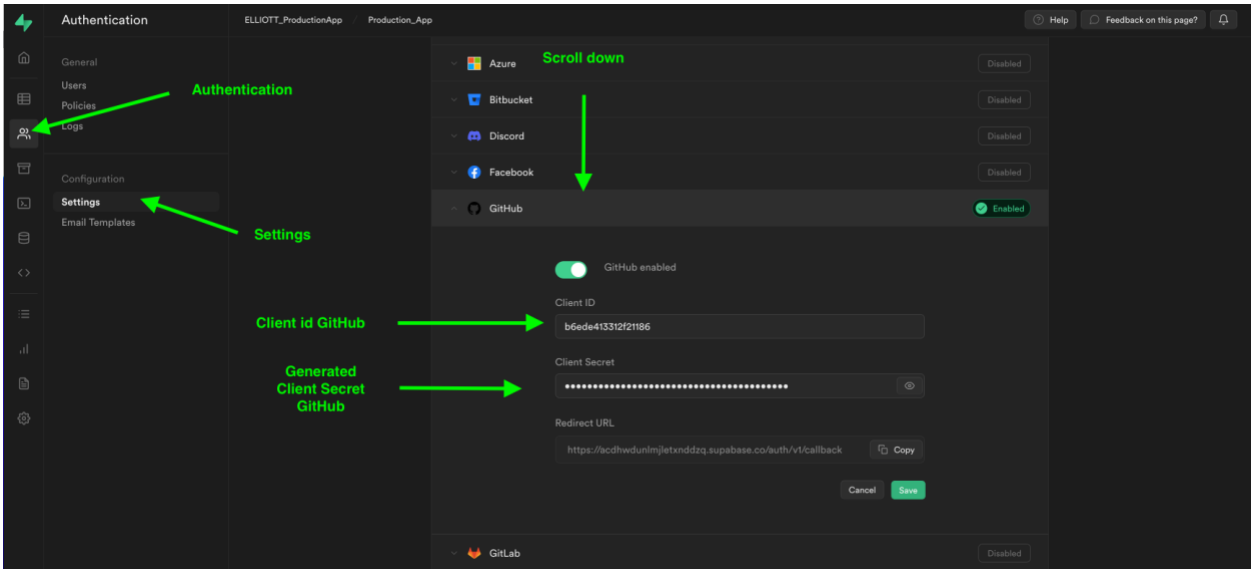
SUPABASE GITHUB AUTH PROVIDER (GITHUB SETUP)

- In your GitHub profile, on the right side of the nav bar, go to Settings / Developer Settings / OAuth apps
- Register a New OAuth application
- Name: your choice
- Homepage URL = Supabase URL
- Authorization Callback = Supabase URL + / auth/v1/callback



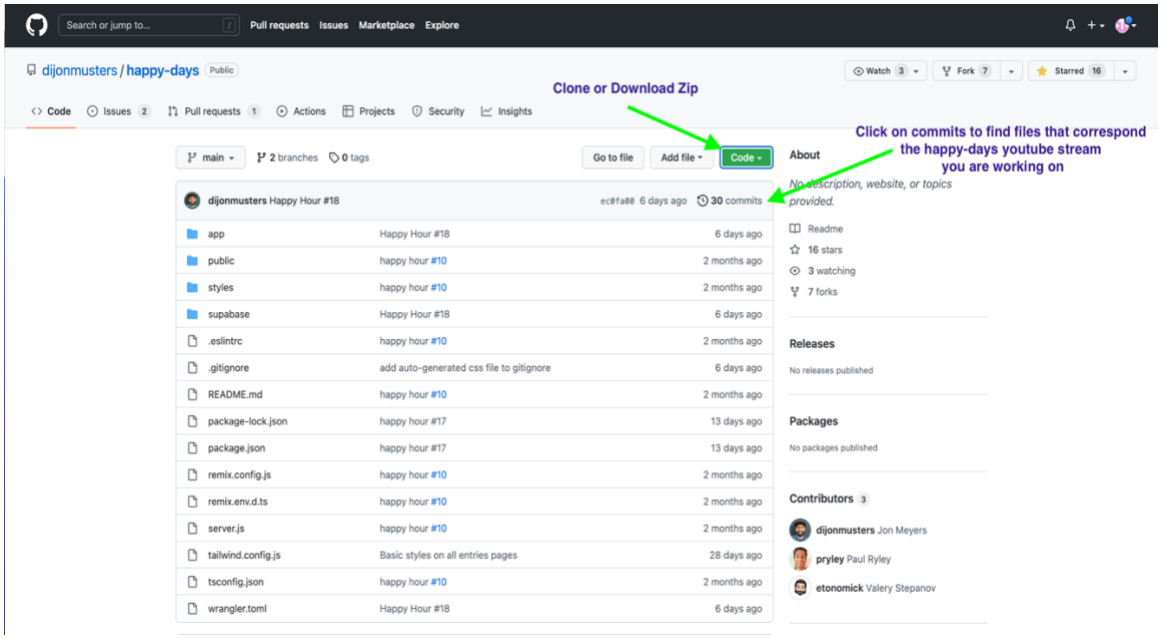
SUPABASE GITHUB AUTH PROVIDER (SUPABASE SETUP)

- After setting up GitHub OAuth copy Client ID AND copy the generated client secrete
- Got to Supabase Dashboard and got to pages Supabase/Authentication/Settings/ and scroll down to Auth Providers
- Enable GitHub, Paste in Client ID, and Client secret and save



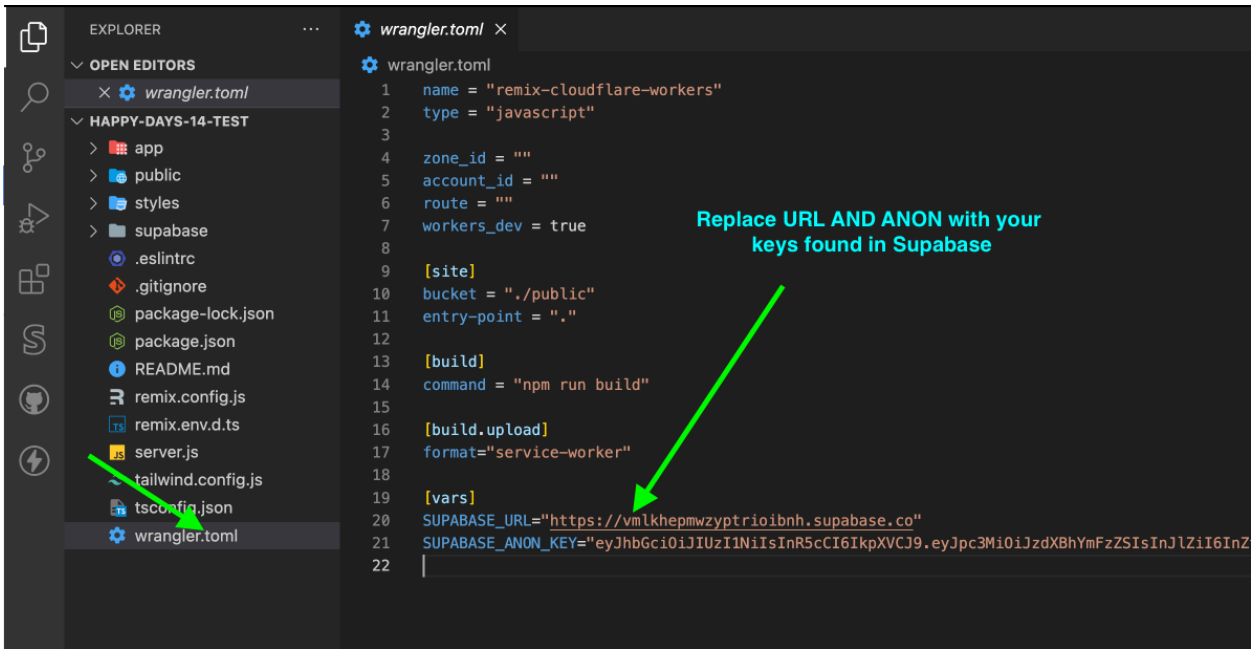
CLONING PROJECT FORM GITHUB

- https://github.com/dijonmusters/happy-days



OPEN IN VSCODE

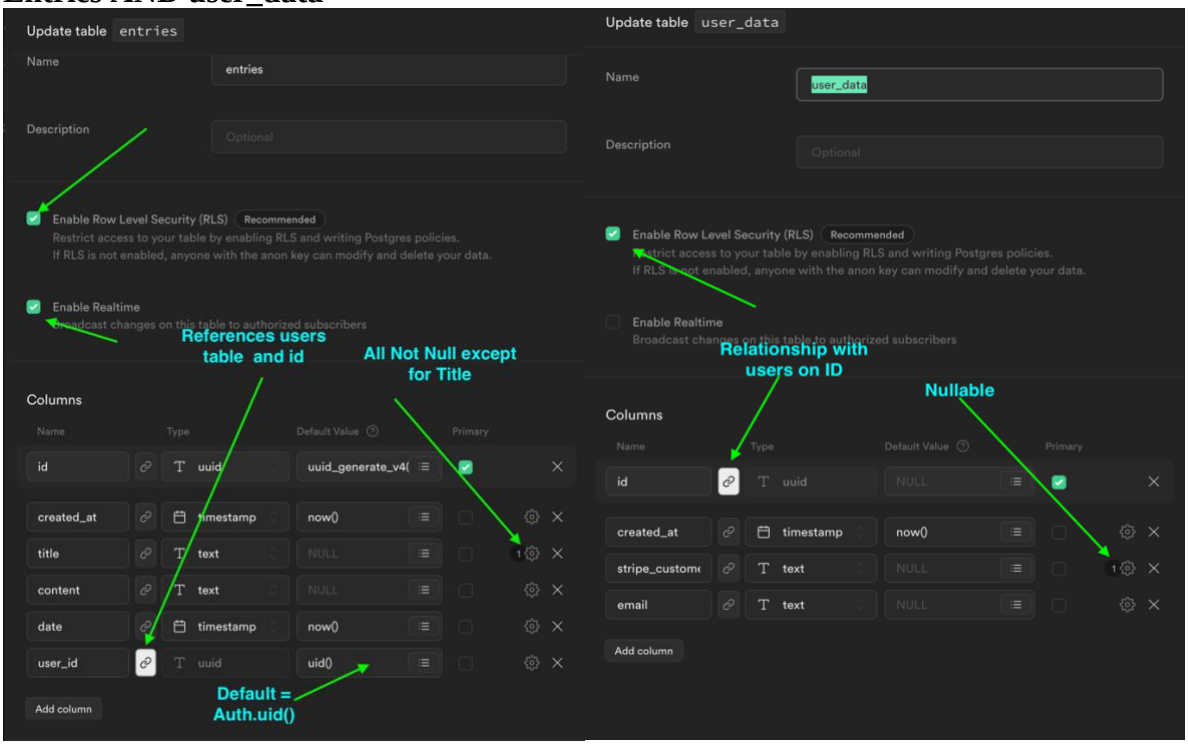
- In wrangler.toml under vars replace SUPABASE_URL and SUPABASE_ANON_KEY with your superbase keys found in the dashboard.
- Continue with VS code and remix after the database is configured.



SETTING UP DATABASE

- Create an entries table and user_data table (Manually or with the SQL code bellow)

Entries AND user_data



SETUP SUPABASE WITH THE SQL CODE BELLOW:(entries,user_data)

```
create table entries (
id uuid default uuid_generate_v4() primary key,
created_at timestamp default now() not null,
title text,
content  text not null,
date timestamp default now() not null,
user_id uuid default auth.uid() references auth.users not null
);

create table user_data(
id uuid references auth.users primary key,
created_at timestamp default now() not null,
stripe_customer_id text,
email text not null
);

alter table entries
enable row level security;

CREATE POLICY "Authenticated users can see their own entries" ON "public"."entries"
AS PERMISSIVE FOR SELECT
TO authenticated
USING (user_id = auth.uid());

CREATE POLICY "users can update their entry" ON "public"."entries"
AS PERMISSIVE FOR UPDATE
TO authenticated
USING (user_id = auth.uid())
WITH CHECK (user_id = auth.uid());

CREATE POLICY "Authenticated users can insert own data" ON "public"."entries"
AS PERMISSIVE FOR INSERT
TO authenticated
WITH CHECK (user_id = auth.uid());


alter table user_data
enable row level security;
```

ENTERING CODE IN SQL EDITOR IN SUPABASE DASHBOARD

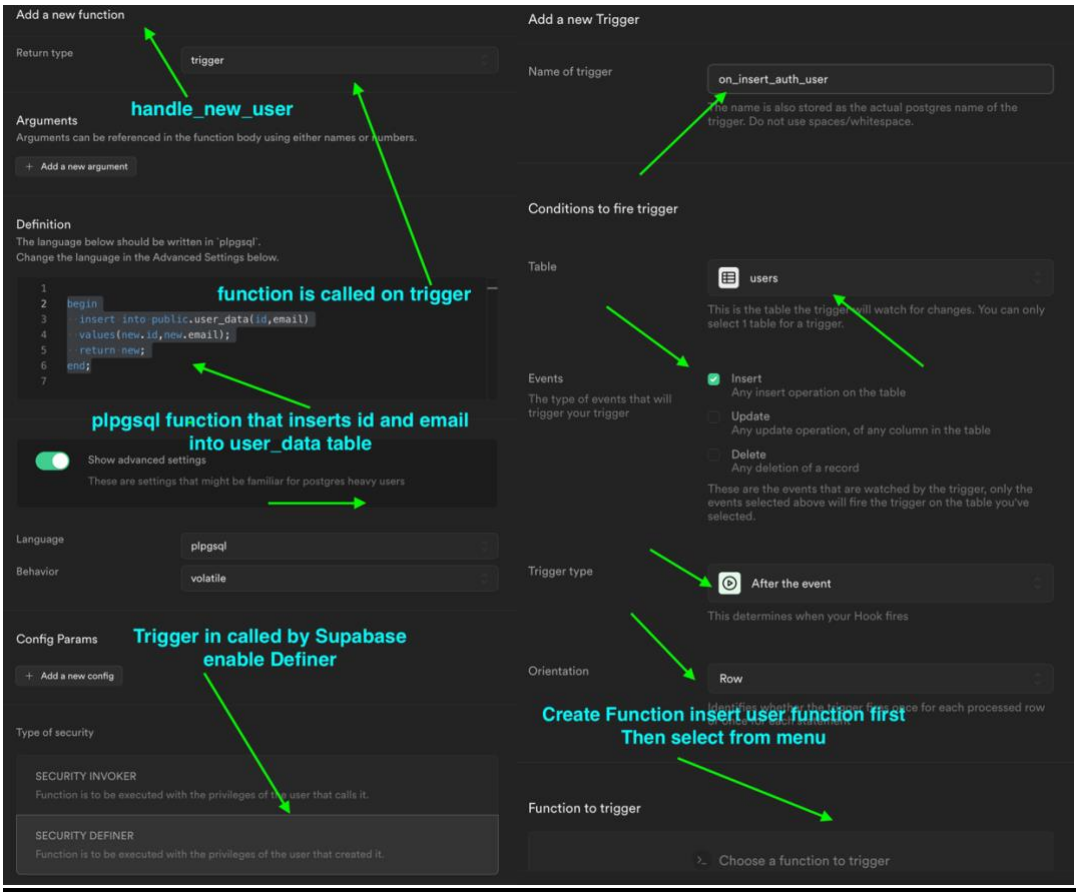
- Paste entries and user_data into SQL code editor

Finally, time to Remix!

- If you have not cloned the repository fallow the directions above.
- Before starting make sure you competed (OAuth, Created tables (entries,user_data, RLS)
- From the cloned repository replace SUPABASE_KEY and Anon_key in wangler.toml.
- RUN npm install to install packages, npm audit fix to install dependencies
- RUN npm run dev to check if OAuth works
- If a user is created the user will show up in the database in supabase/authentication/users.
- If the user is created continue to set up a function, that creates a user in the user_data table, trigged by the login.

SETTING UP FUNCTIONS AND TRIGGERS (MANUALLY)

- In the supabase dashboard go to Supabase/ Database / In the menu below, you will see Triggers / Functions and Database Webhooks.
- Create the handle_new_users function first
- After the function is created you can connect the handle_new_users function to the on_insert_auth_user trigger. To connect the trigger-function use the drop-down menu “functions to trigger” in the trigger and you will see the function handle_new_users.



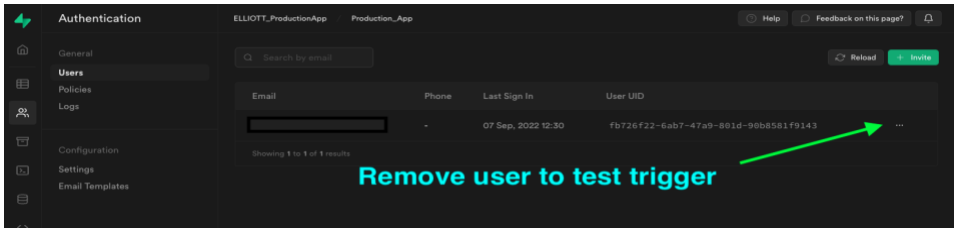
SETTING UP FUNCTION + TIGGER IN SQL EDITOR

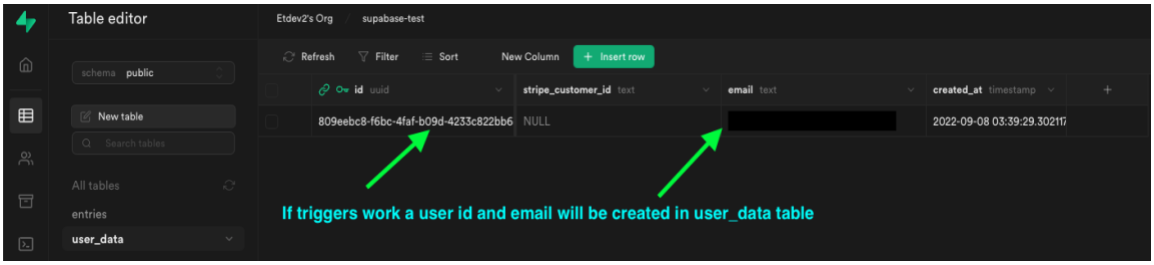
```
create function public.handle_new_user()
returns trigger as
$$
begin
insert into public.user_data(id,email)
values(new.id,new.email);
return new;
end;

$$
language plpgsql security definer;

create trigger on_insert_auth_user
after insert on auth.users
for each row
execute procedure public.handle_new_user();
```

IN VSCODE IN TERMAL

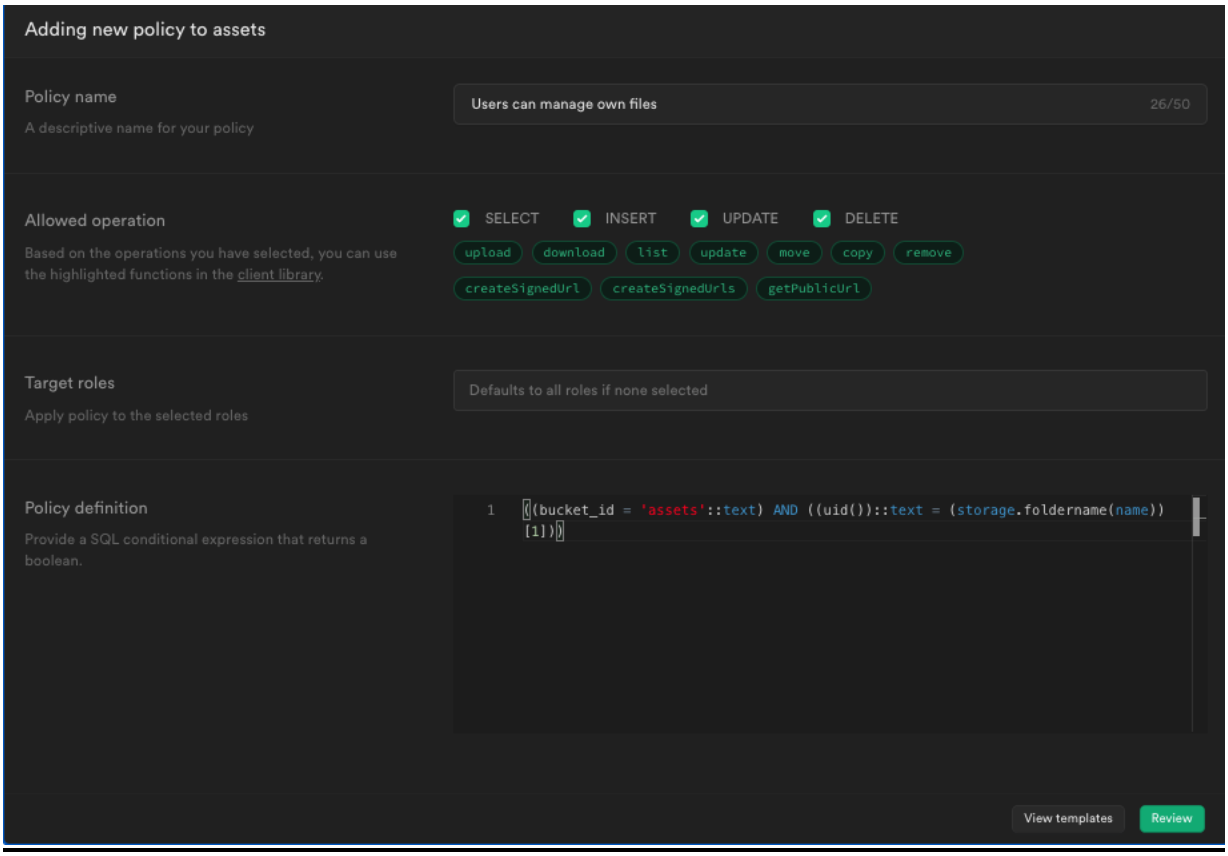
- **REMOVE** all the user data from the supabase including the user before testing the trigge
- 

- **RUN npm run dev**
- If everything is working properly after signing in to the remix app. A user with user_id will be written into the user_data base with stripe_customer null.
- 

SUPABASE EDGE FUNCTIONS AND WEBHOOKS

SETUP STORAGE

- Create a new storage bucket named assets
- Allow all operations
- Added the below definitions
- ((bucket_id = 'assets'::text) AND ((uid())::text = (storage.foldername(name))[1]))



SUPABASE EDGE FUNCTIONS AND SUPABASE CLI (Run Code In Vscode Termal Exclude Run In The Terminal**)**

STEPS TO THE EDGE

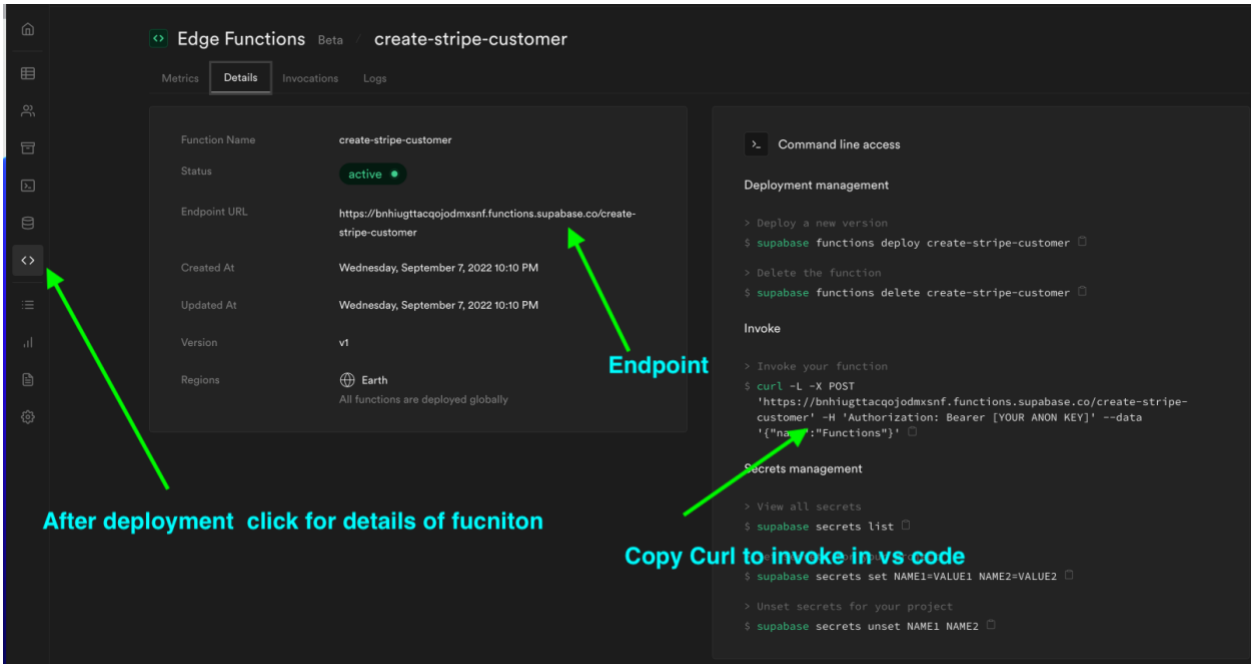
Follow steps below is more detailed may have messed around a few time to get everything working, It took me a few times to make sure everything was right and spelled correctly

1. **Remove supabases file from clone (More detailed below)**

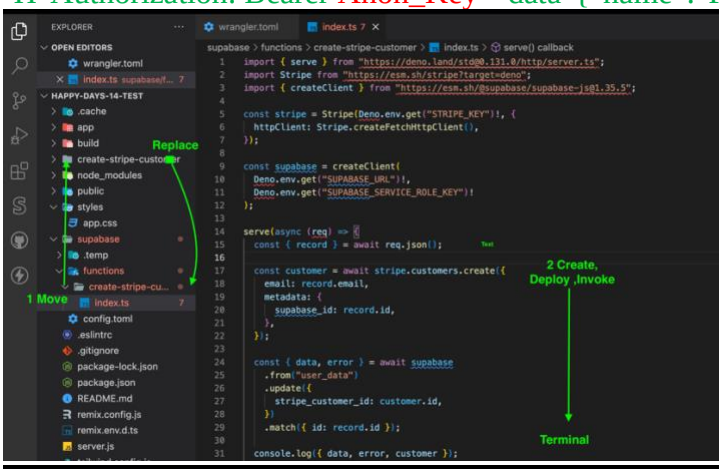
2. **supabase init**
3. **supabase link**
4. **supabase functions new create-stripe-customer**
5. **supabase functions deploy create-stripe-customer**
6. **curl -L -X POST invoke the function**
7. **Create database webhook in supabase**
8. **Npm run dev and login**
9. **If user is found in user_data with stripe customer null everythning is working continue and remove all user data from auth and user_data table**
10. **Kill server**
11. **supabase secretes list**
12. **supabase secrets set STRIPE_KEY = sk_13234**
13. **supabase secretes list again and check**
14. **Copy code from code in supabase/functions/index from Happys-14 repo (the first function on top of index starts with const stripe = stripe(Deno.env.get....)**
15. **supabase functions deploy create-stripe-customer**
16. **Finally; npm run dev, start server, login. If no errors everything probably works, check in supabase dashboard in the user_data table to see if the user was created with a stripe_customer_id, if yes, then a customer will be inserted into the stripe dashboard.**
17. **(BELOW IS MORE DETAILED)(FOLLOW THE STEPS)**

- **https://supabase.com/docs/guides/cli**
- Move temperately create-stripe-customer out of the superbase file before creating edge function.
- RUN supabase INIT

- RUN supabase link --project-ref [Project-ref]
- After linking project you can excluded your project ref when deploying edge function
- RUN supabase functions new create-stripe-customer (Creates)
- RUN supabase functions deploy create-stripe-customer --project-ref [project-ref] (deploys)

- 

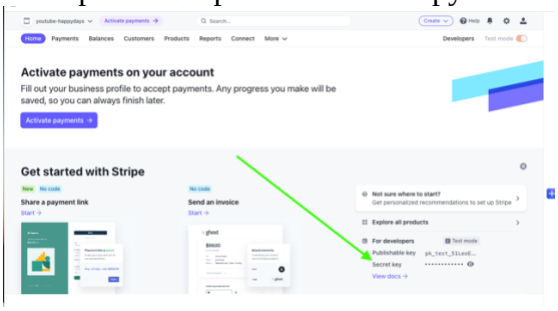
- RUN curl -L -X POST 'https:// [project-ref].functions.supabase.co/create-stripe-customer' -H 'Authorization: Bearer Anon_Key --data '{"name":"Functions"}'(Invokes)



- Replace the create-stripe-customer folder, that was moved earlier in vs code

CREATE A STRIPE SECRETE KEY THAT CAN BE CALLED BY FUNCTION

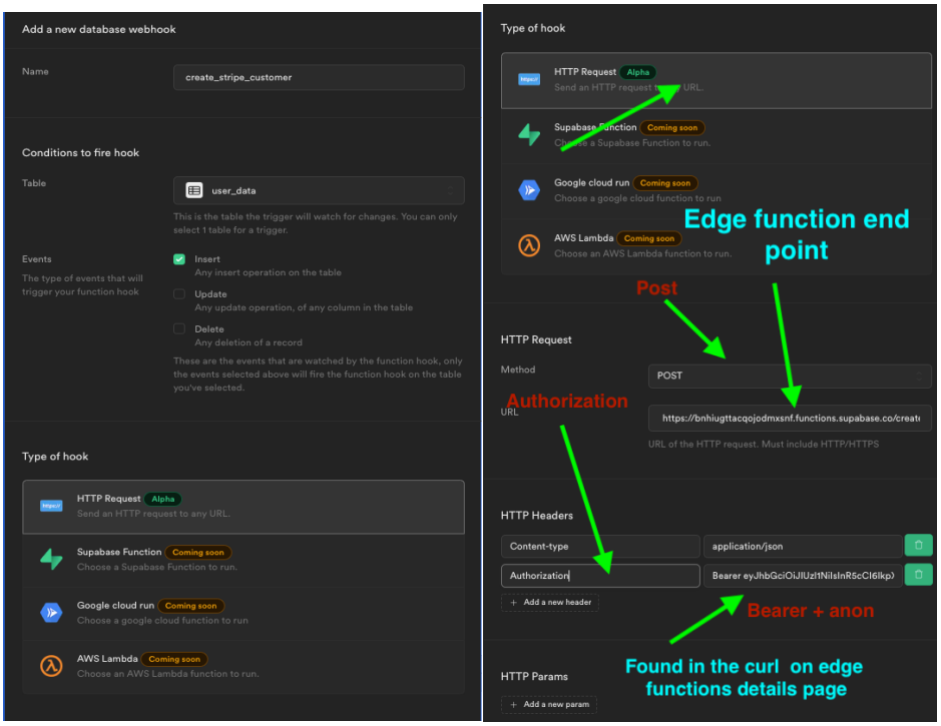
- Create a stripe account
- In stripes developer dashboard copy secrete key



- In vscode terminal Run: supabase secrets list (get a list of keys stored in supabase)
- To set stripe key in Supabase Run: supabase secrets set STRIPE_KEY =sk_1234 in terminal
- Run: supabase secrets list to check if stripe key was stored

CREATE A SUPABASE DATABASE WEBHOOK

- Create a webhook to call the Edge function that will create a stripe customer in stripe and in supabase when a new user is inserted into the database.
- Delete user_data run npm run dev, login and if everything works you should have created a stripe customer in user_data and stripe dash board



ADD A NEW PRODUCT IN STRIPE

- **Free/standard/premium**

- **RECURRING MONTHLY**
- **$0 / $4.99/ $9.99**

To be continued…. This should work up to happy hour #15 more testing is needed, which will save you time! You will need to mess around with things to get it to work properly

SQL Should bring database tables to and a trigger + function you need to add database webhooks and supabase Edge functions

Run first

```
create type subscription_tier_test as enum('FREE','STADARD','PREMIUM')
```

```
create table entries (
  id uuid default uuid_generate_v4() primary key,
  created_at timestamp default now() not null,
  title text,
  content  text not null,
  date timestamp default now() not null,
  user_id uuid default auth.uid() references auth.users not null,
  asset_urls text []  not null
);


create table user_data
 (
    id uuid references auth.users primary key,
    created_at timestamp default now() not null,
    stripe_customer_id text,
    email text not null,
    subscription_tier   subscription_tier default 'FREE' not null
 );


alter table entries
  enable row level security;
```

```sql
CREATE POLICY "Authenticated users can see their own entries" ON "public"."entries"
AS PERMISSIVE FOR SELECT
TO authenticated
USING (user_id = auth.uid());

CREATE POLICY "users can update their entry" ON "public"."entries"
AS PERMISSIVE FOR UPDATE
TO authenticated
USING (user_id = auth.uid())
WITH CHECK (user_id = auth.uid());

CREATE POLICY "Authenticated users can insert own data" ON "public"."entries"
AS PERMISSIVE FOR INSERT
TO authenticated
WITH CHECK (user_id = auth.uid());


alter table user_data
  enable row level security;

create function public.handle_new_user()
returns trigger as
$$
begin
  insert into public.user_data(id,email)
  values(new.id,new.email);
  return new;
end;

$$
language plpgsql security definer;
```

```
create trigger on_insert_auth_user
  after insert on auth.users
  for each row
    execute procedure public.handle_new_user();
```