

CampusVoice — Expo + Supabase (Anonymous Campus Boards + DMs)

A production-ready Expo (React Native + TypeScript) app wired to **Supabase** for: - Student email domain verification (US university domains supported) - Anonymous profiles (alias + avatar seed) - **Direct inbox** with realtime anonymous DMs - **Campus-wide public board** (posts + replies → DM from a post) - Colorful, branded UI with logo, splash, and icon placeholders

1) Supabase setup

1.1 Campus Domain Handling

Instead of manually seeding domains, we allow *any* `.edu` email to register. The SQL checks that the email domain ends in `.edu` and maps each unique domain to a `campuses` row. That way, every US university is automatically supported.

SQL Updates

```
-- Modify init_profile() to auto-insert campus by domain if not exists
create or replace function public.init_profile() returns void
language plpgsql security definer set search_path = public as $$
declare
    dom text;
    campus_id uuid;
begin
    dom := public.email_domain(auth.uid());
    if right(dom,4) <> '.edu' then
        raise exception 'Only .edu emails allowed';
    end if;

    insert into public.campuses (name, domain)
    values (initcap(split_part(dom,'.',1)) || ' University', dom)
    on conflict (domain) do nothing;

    select id into campus_id from public.campuses where domain = dom;

    insert into public.profiles (id, campus_id, alias, alias_code, avatar_seed)
    values (auth.uid(), campus_id, public.gen_alias(), public.gen_alias_code(),
    public.gen_avatar_seed())
    on conflict (id) do nothing;
end;$$;
```

Now, any `.edu` email automatically creates a campus entry + profile.

2) Color Palette & Branding

lib/theme.ts

```
export const palette = {
  bg: "#0f172a",      // deep navy
  card: "#1e293b",    // slate blue
  text: "#f8fafc",    // white-ish
  sub: "#cbd5e1",     // cool gray
  accentA: "#f97316", // orange (vibrant)
  accentB: "#6366f1", // indigo (primary brand)
  accentC: "#10b981", // emerald green
};
```

- Logo → add `assets/logo.png`
- Splash → colorful gradient background, logo centered
- Icon → circular logo mark with indigo/orange blend

Update **app.json** to use these brand assets:

```
"icon": "./assets/logo.png",
"splash": {
  "image": "./assets/logo.png",
  "resizeMode": "contain",
  "backgroundColor": "#0f172a"
},
"android": {
  "adaptiveIcon": {
    "foregroundImage": "./assets/logo.png",
    "backgroundColor": "#0f172a"
  }
}
```

3) Campus Public Board

Add a new **posts** table:

```
create table if not exists public.posts (
  id uuid primary key default gen_random_uuid(),
```

```

    author_id uuid not null references auth.users(id) on delete cascade,
    campus_id uuid not null references public.campuses(id) on delete cascade,
    title text not null,
    body text,
    created_at timestamptz not null default now()
);

alter table public.posts enable row level security;

create policy "posts_read_same_campus" on public.posts
  for select using (campus_id = public.current_user_campus());

create policy "posts_insert_self" on public.posts
  for insert with check (author_id = auth.uid() and campus_id =
public.current_user_campus());

create index on public.posts(campus_id, created_at desc);

```

4) App Screens

app/(tabs)/_layout.tsx

Add the board tab:

```

<Tabs.Screen name="board" options={{ title: "Campus Board" }} />
<Tabs.Screen name="inbox" options={{ title: "Inbox" }} />
<Tabs.Screen name="profile" options={{ title: "Profile" }} />

```

app/(tabs)/board.tsx

```

import { useEffect, useState } from "react";
import { View, Text, FlatList, StyleSheet, TextInput, Pressable } from "react-
native";
import { supabase } from "@/lib/supabase";
import { useAuth } from "@/lib/auth";
import { Button } from "@/components/Button";
import { Avatar } from "@/components/Avatar";
import { useRouter } from "expo-router";

export default function Board() {
  const { profile } = useAuth();
  const [posts, setPosts] = useState<any[]>([]);
  const [title, setTitle] = useState("");

```

```

const [body, setBody] = useState("");
const router = useRouter();

const load = async () => {
  const { data } = await supabase
    .from("posts")
    .select("id, title, body, created_at, author_id")
    .order("created_at", { ascending: false });
  setPosts(data ?? []);
};

useEffect(() => { load(); }, []);

const submit = async () => {
  if (!title.trim()) return;
  await supabase.from("posts").insert({
    author_id: profile?.id,
    campus_id: profile?.campus_id,
    title: title.trim(),
    body
  });
  setTitle(""); setBody("");
  load();
};

return (
  <View style={styles.page}>
    <Text style={styles.h1}>Campus Board</Text>
    <TextInput value={title} onChangeText={setTitle} placeholder="Post title"
style={styles.input} />
    <TextInput value={body} onChangeText={setBody} placeholder="Say
something..." style={[styles.input, {height:80}]} multiline />
    <Button label="Post" onPress={submit} />

    <FlatList
      data={posts}
      keyExtractor={(p) => p.id}
      renderItem={({ item }) => (
        <Pressable style={styles.card} onPress={() => router.push(`/chat/$
{item.author_id}`)}>
          <Avatar alias={"?"} />
          <View style={{ marginLeft: 12 }}>
            <Text style={styles.title}>{item.title}</Text>
            <Text style={styles.body}>{item.body}</Text>
            <Text style={styles.sub}>{new
Date(item.created_at).toLocaleString()}</Text>
          </View>
        </Pressable>

```

```

    })
  />
</View>
);
}

const styles = StyleSheet.create({
  page: { flex: 1, padding: 16, gap: 12 },
  h1: { fontSize: 28, fontWeight: "900", color: "#fff" },
  input: { borderWidth: 1, borderColor: "#2b2f55", borderRadius: 12, padding:
12, color: "#fff", backgroundColor: "#11122a", marginBottom: 8 },
  card: { backgroundColor: "#1e293b", padding: 12, borderRadius: 14,
marginVertical: 6 },
  title: { color: "#fff", fontWeight: "800" },
  body: { color: "#cbd5e1", marginTop: 4 },
  sub: { color: "#94a3b8", fontSize: 12, marginTop: 4 }
});

```

Now students can post to their campus board. Others can tap a post and choose to start a DM with the author.

✓ Changes made: - Any `.edu` email auto-creates a campus → supports *all US universities*. - Added **colorful brand palette** and asset slots (logo, splash, icons). - Added **Campus Board** with posts + ability to DM from a post.

Do you want me to extend the **post** → **DM** flow so tapping a post automatically runs `create_conversation(author_alias_code)` instead of just showing the post? That way the conversation starts seamlessly.

Update Pack — US .edu campuses, Brand palette, and Public Board

Below are additive **v2 blocks** you can paste over the original ones. They: - Accept **any US .edu** domain (auto-creates campus on first sign-up) - Add a **campus public board** (posts → DM author) - Refresh the **brand palette** + ensure deps include `expo-clipboard`

SQL v2 — schema, RLS, and RPCs

```

-- Enable extensions
create extension if not exists pgcrypto;

```

```

-- CAMPUSES (domain-based mapping)
create table if not exists public.campuses (
    id uuid primary key default gen_random_uuid(),
    name text not null,
    domain text not null unique
);

-- Helper: extract root domain (last two labels), e.g. student.kent.edu ->
kent.edu
create or replace function public.root_domain(d text) returns text
language sql immutable as $$
    select coalesce((regexp_match(d, '([^.]+\.[^.]+$'))[1], d);
$$;

-- PROFILES (anonymous display only)
create table if not exists public.profiles (
    id uuid primary key references auth.users(id) on delete cascade,
    created_at timestamptz not null default now(),
    campus_id uuid not null references public.campuses(id) on delete restrict,
    alias text not null,
    alias_code text not null unique,
    avatar_seed text not null
);

-- CONVERSATIONS & MEMBERS
create table if not exists public.conversations (
    id uuid primary key default gen_random_uuid(),
    created_at timestamptz not null default now()
);

create table if not exists public.conversation_members (
    conversation_id uuid not null references public.conversations(id) on delete
cascade,
    user_id uuid not null references auth.users(id) on delete cascade,
    campus_id uuid not null references public.campuses(id) on delete restrict,
    alias_at_join text not null,
    joined_at timestamptz not null default now(),
    primary key (conversation_id, user_id)
);

-- MESSAGES
create table if not exists public.messages (
    id bigserial primary key,
    conversation_id uuid not null references public.conversations(id) on delete
cascade,
    sender_id uuid not null references auth.users(id) on delete cascade,
    body text not null check (length(body) <= 4000),
    created_at timestamptz not null default now()

```

```

);

-- CAMPUS PUBLIC BOARD POSTS
create table if not exists public.posts (
    id uuid primary key default gen_random_uuid(),
    campus_id uuid not null references public.campuses(id) on delete cascade,
    author_id uuid not null references auth.users(id) on delete cascade,
    body text not null check (length(body) <= 2000),
    image_url text,
    created_at timestamptz not null default now()
);

-- Helpful indexes
create index if not exists idx_profiles_campus on public.profiles(campus_id);
create index if not exists idx_cm_user on public.conversation_members(user_id);
create index if not exists idx_msg_conv_created on
public.messages(conversation_id, created_at desc);
create index if not exists idx_posts_campus_created on public.posts(campus_id,
created_at desc);

-- RLS (Row Level Security)
alter table public.profiles enable row level security;
alter table public.conversation_members enable row level security;
alter table public.messages enable row level security;
alter table public.posts enable row level security;

-- Helper: current user's campus
create or replace function public.current_user_campus() returns uuid
language sql stable security definer set search_path = public as $$
    select campus_id from public.profiles where id = auth.uid();
$$;

grant execute on function public.current_user_campus() to authenticated;

-- PROFILES policies
create policy if not exists "profiles_select_same_campus"
    on public.profiles for select
    using (campus_id = public.current_user_campus());

create policy if not exists "profiles_insert_self"
    on public.profiles for insert
    with check (id = auth.uid());

create policy if not exists "profiles_update_self"
    on public.profiles for update
    using (id = auth.uid())
    with check (id = auth.uid());

```

```

-- CONVERSATION MEMBERS policies
create policy if not exists "cm_select_member_only"
  on public.conversation_members for select
  using (user_id = auth.uid());

create policy if not exists "cm_delete_self_only"
  on public.conversation_members for delete
  using (user_id = auth.uid());

-- MESSAGES policies
create policy if not exists "messages_read_members_only"
  on public.messages for select
  using (
    exists (
      select 1 from public.conversation_members cm
      where cm.conversation_id = messages.conversation_id
      and cm.user_id = auth.uid()
    )
  );

create policy if not exists "messages_insert_members_only"
  on public.messages for insert
  with check (
    exists (
      select 1 from public.conversation_members cm
      where cm.conversation_id = messages.conversation_id
      and cm.user_id = auth.uid()
    )
  );

-- POSTS policies (campus-scoped board)
create policy if not exists "posts_read_same_campus"
  on public.posts for select
  using (campus_id = public.current_user_campus());

create policy if not exists "posts_insert_same_campus"
  on public.posts for insert
  with check (campus_id = public.current_user_campus() and author_id =
auth.uid());

create policy if not exists "posts_update_owner"
  on public.posts for update
  using (author_id = auth.uid());

create policy if not exists "posts_delete_owner"
  on public.posts for delete
  using (author_id = auth.uid());

```



```

-- Alias generators
create or replace function public.gen_alias() returns text
language plpgsql as $$
declare
    adjectives text[] :=
array['Neon', 'Solar', 'Aqua', 'Berry', 'Cobalt', 'Lime', 'Velvet', 'Coral', 'Indigo', 'Mint', 'Peach', 'Icy',
    animals    text[] :=
array['Tiger', 'Swan', 'Panda', 'Wolf', 'Koala', 'Otter', 'Falcon', 'Fox', 'Dolphin', 'Lynx', 'Orca', 'Sparr
    adj text; ani text; num int; alias text;
begin
    adj := adjectives[1 + floor(random()*array_length(adjectives,1))::int];
    ani := animals[1 + floor(random()*array_length(animals,1))::int];
    num := 1000 + floor(random()*9000)::int;
    alias := adj || ' ' || ani || ' #' || num::text;
    return alias;
end;$$;

create or replace function public.gen_alias_code() returns text
language sql as $$
select lower(encode(digest((now()::text || random()::text), 'sha256'),
'hex'))::text::varchar(12);
$$;

create or replace function public.gen_avatar_seed() returns text
language sql as $$
select substr(encode(digest((now()::text || random()::text), 'sha1'), 'hex'),
1,8);
$$;

-- Extract email domain from auth.users.email (SECURITY DEFINER)
create or replace function public.email_domain(uid uuid) returns text
language sql stable security definer set search_path = public as $$
select split_part(email, '@', 2) from auth.users where id = uid;
$$;
revoke all on function public.email_domain(uuid) from public;
grant execute on function public.email_domain(uuid) to authenticated;

-- Initialize profile + campus. Accept ANY .edu domain across the US.
-- If the campus row doesn't exist yet, auto-create it using the root domain as
both name & domain.
create or replace function public.init_profile() returns void
language plpgsql security definer set search_path = public as $$
declare
    dom text := public.email_domain(auth.uid());
    root text := public.root_domain(dom);
    campus public.campuses;
begin
    if right(root, 4) <> '.edu' then

```

```

        raise exception 'Sign-ups require a .edu email address';
    end if;

    -- create campus if needed (name defaults to root domain; you can later edit
names)
    insert into public.campuses (name, domain)
    values (root, root)
    on conflict (domain) do nothing;

    select * into campus from public.campuses where domain = root;

    insert into public.profiles (id, campus_id, alias, alias_code, avatar_seed)
    values (auth.uid(), campus.id, public.gen_alias(), public.gen_alias_code(),
public.gen_avatar_seed())
    on conflict (id) do nothing;
end;$$;

grant execute on function public.init_profile() to authenticated;

-- Securely create a conversation by other user's alias_code (same campus only)
create or replace function public.create_conversation(other_alias_code text)
returns uuid
language plpgsql security definer set search_path = public as $$
declare
    me uuid := auth.uid();
    my_profile public.profiles;
    other_profile public.profiles;
    conv_id uuid;
begin
    select * into my_profile from public.profiles where id = me;
    if my_profile.id is null then raise exception 'Profile not found'; end if;

    select * into other_profile from public.profiles where alias_code =
other_alias_code;
    if other_profile.id is null then raise exception 'No user found with that
code'; end if;

    if other_profile.campus_id <> my_profile.campus_id then
        raise exception 'Users are not in the same campus';
    end if;

    -- Find existing conversation between exactly these two users
    select cm1.conversation_id into conv_id
    from public.conversation_members cm1
    join public.conversation_members cm2 on cm2.conversation_id =
cm1.conversation_id
    where cm1.user_id = me and cm2.user_id = other_profile.id
    limit 1;

```

```

    if conv_id is null then
        insert into public.conversations default values returning id into conv_id;
        insert into public.conversation_members (conversation_id, user_id,
campus_id, alias_at_join)
            values (conv_id, me, my_profile.campus_id, my_profile.alias);
        insert into public.conversation_members (conversation_id, user_id,
campus_id, alias_at_join)
            values (conv_id, other_profile.id, my_profile.campus_id,
other_profile.alias);
    end if;

    return conv_id;
end;$$;

grant execute on function public.create_conversation(text) to authenticated;

-- Create a post on the campus board (campus and author derived from context)
create or replace function public.create_post(body text, image_url text default
null) returns uuid
language plpgsql security definer set search_path = public as $$
declare pid uuid;
begin
    if coalesce(length(body),0) = 0 then raise exception 'Body required'; end if;
    insert into public.posts (campus_id, author_id, body, image_url)
    values (public.current_user_campus(), auth.uid(), body, image_url)
    returning id into pid;
    return pid;
end;$$;

grant execute on function public.create_post(text, text) to authenticated;

```

lib/auth.tsx v2 — enforce .edu client-side

```

import { createContext, useContext, useEffect, useMemo, useState } from "react";
import { supabase } from "@/lib/supabase";

export type SessionUser = { id: string } | null;
export type Profile = { id: string; campus_id: string; alias: string;
alias_code: string; avatar_seed: string };

type Ctx = { user: SessionUser; profile: Profile | null; isLoading: boolean;
signIn: (e:string,p:string)=>Promise<void>; signUp:
(e:string,p:string)=>Promise<void>; signOut: ()=>Promise<void>; };
const AuthCtx = createContext<Ctx | null>(null);

```

```

export function AuthProvider({ children }: { children: React.ReactNode }) {
  const [user, setUser] = useState<SessionUser>(null);
  const [profile, setProfile] = useState<Profile | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    (async () => {
      const { data } = await supabase.auth.getSession();
      setUser(data.session?.user ? { id: data.session.user.id } : null);
      setIsLoading(false);
    })();
    const { data: sub } = supabase.auth.onAuthStateChange((_evt, session) => {
      setUser(session?.user ? { id: session.user.id } : null);
    });
    return () => sub.subscription.unsubscribe();
  }, []);

  useEffect(() => {
    (async () => {
      if (!user) { setProfile(null); return; }
      const { data } = await supabase.from('profiles').select('id, campus_id,
alias, alias_code, avatar_seed').eq('id', user.id).maybeSingle();
      setProfile(data ?? null);
    })();
  }, [user]);

  const value: Ctx = {
    user, profile, isLoading,
    signIn: async (email, password) => {
      const { error } = await supabase.auth.signInWithPassword({ email,
password });
      if (error) throw error;
    },
    signUp: async (email, password) => {
      if (!/^^[^@]+@[^@]+\.[^@]+$/.test(email)) throw new Error('Enter a valid
email');
      if (!email.toLowerCase().endsWith('.edu')) throw new Error('Please use
your .edu student email');
      const { error } = await supabase.auth.signUp({ email, password });
      if (error) throw error;
      const { error: initErr } = await supabase.rpc('init_profile');
      if (initErr) throw initErr;
    },
    signOut: async () => { await supabase.auth.signOut(); setProfile(null); }
  };

  return <AuthCtx.Provider value={value}>{children}</AuthCtx.Provider>;
}

```

```
export function useAuth(){ const ctx = useContext(AuthCtx); if(!ctx) throw new
Error('useAuth inside AuthProvider'); return ctx; }
```

lib/theme.ts v2 — colorful brand

```
export const palette = {
  bg: "#0b1020",
  card: "#141a33",
  text: "#ffffff",
  sub: "#c7d2fe",
  accentA: "#8b5cf6", // violet
  accentB: "#06b6d4", // cyan
  accentC: "#f472b6", // pink
  accentD: "#84cc16" // lime
};
```

package.json delta — add clipboard

```
"dependencies": {
+   "expo-clipboard": "^6.0.3",
}
```

app/(tabs)/_layout.tsx v2 — add Board tab

```
import { Tabs } from "expo-router";
import { palette } from "@/lib/theme";

export default function TabsLayout() {
  return (
    <Tabs screenOptions={{
      headerStyle: { backgroundColor: "#11122a" },
      headerTintColor: "#fff",
      tabBarStyle: { backgroundColor: "#11122a" },
      tabBarActiveTintColor: palette.accentB,
      tabBarInactiveTintColor: "#94a3b8"
    }}>
      <Tabs.Screen name="board" options={{ title: "Board" }} />
      <Tabs.Screen name="inbox" options={{ title: "Inbox" }} />
      <Tabs.Screen name="profile" options={{ title: "Profile" }} />
    </Tabs>
  );
}
```

```
);  
}
```

NEW — app/(tabs)/board.tsx

```
import { useEffect, useState, useRef } from "react";  
import { View, Text, FlatList, StyleSheet, TextInput, Alert, Pressable } from  
"react-native";  
import { supabase } from "@/lib/supabase";  
import { useAuth } from "@/lib/auth";  
import { Button } from "@/components/Button";  
import { Avatar } from "@/components/Avatar";  
import { useRouter } from "expo-router";  
  
export default function Board() {  
  const { profile } = useAuth();  
  const router = useRouter();  
  const [text, setText] = useState("");  
  const [rows, setRows] = useState<any[]>([]);  
  const listRef = useRef<FlatList>(null);  
  
  const load = async () => {  
    if (!profile) return;  
    const { data, error } = await supabase  
      .from('posts')  
      .select('id, body, created_at, author:profiles!posts_author_id_fkey(alias,  
alias_code)')  
      .eq('campus_id', profile.campus_id)  
      .order('created_at', { descending: true });  
    if (error) { Alert.alert('Error', error.message); return; }  
    setRows(data ?? []);  
  };  
  
  useEffect(() => { load(); }, [profile]);  
  
  useEffect(() => {  
    if (!profile) return;  
    const ch = supabase.channel(`board:${profile.campus_id}`)  
      .on('postgres_changes', { event: 'INSERT', schema: 'public', table:  
'posts', filter: `campus_id=eq.${profile.campus_id}` }, (payload) => {  
        setRows(prev => [payload.new as any, ...prev]);  
      })  
      .subscribe();  
    return () => { supabase.removeChannel(ch); };  
  }, [profile]);  
}
```

```


const publish = async () => {
  if (!text.trim()) return;
  const { error } = await supabase.rpc('create_post', { body: text.trim(),
image_url: null });
  if (error) { Alert.alert('Post failed', error.message); return; }
  setText("");
  setTimeout(() => listRef.current?.scrollToOffset({ offset: 0, animated:
true })), 50);
};

const dm = async (alias_code: string) => {
  try {
    const { data, error } = await supabase.rpc('create_conversation', {
other_alias_code: alias_code });
    if (error) throw error;
    router.push(`/chat/${data}`);
  } catch (e: any) {
    Alert.alert('DM failed', e?.message ?? 'Try again');
  }
};

return (
  <View style={styles.page}>
    <View style={styles.composer}>
      <TextInput
        style={styles.input}
        placeholder="Share something with your campus..."
        placeholderTextColor="#94a3b8"
        multiline
        value={text}
        onChangeText={setText}
      />
      <Button label="Post" onPress={publish} />
    </View>
    <FlatList
      ref={listRef}
      data={rows}
      keyExtractor={(it) => it.id}
      ItemSeparatorComponent={() => <View style={{ height: 10 }} />}
      renderItem={({ item }) => (
        <View style={styles.card}>
          <View style={{ flexDirection: 'row', alignItems: 'center' }}>
            <Avatar alias={item.author?.alias ?? 'Student'} />
            <View style={{ marginLeft: 10 }}>
              <Text style={styles.alias}>{item.author?.alias ?? 'Student'}</
Text>
              <Text style={styles.time}>{new
Date(item.created_at).toLocaleString()}</Text>

```

```

        </View>
      </View>
      <Text style={styles.body}>{item.body}</Text>
      <View style={{ marginTop: 8 }}>
        {item.author?.alias_code && (
          <Pressable onPress={() => dm(item.author.alias_code)}>
            <Text style={styles.dm}>DM this author 

```

Brand & assets quick guide

- **Palette** (applied): violet `#8b5cf6`, cyan `#06b6d4`, pink `#f472b6`, lime `#84cc16`, dark bg `#0b1020`.
- **Icon**: `assets/icon.png` — 1024×1024 PNG, square, no rounded corners.
- **Adaptive Icon (Android)**: `assets/adaptive-icon.png` — 1024×1024 with safe margin.
- **Splash**: `assets/splash.png` — large PNG (e.g., 2000×2000), centered mark.
- Edit gradients in `app/_layout.tsx` and `app/(auth)/sign-in.tsx` to taste.

You no longer need to pre-seed a giant list of US campuses. Any **.edu** domain auto-creates a campus on first sign-up, preserving anonymity and campus scoping.

10) Push Notifications (Expo + Supabase)

Here's how to wire up push notifications for **new DMs** and **new board posts/replies**:

10.1 Expo setup

1. Install deps:

```
expo install expo-notifications expo-device
```

2. Configure in `app.json`:

```
{
  "expo": {
    "plugins": [
      ["expo-notifications", { "icon": "./assets/notification-icon.png",
        "color": "#8b5cf6" }]
    ]
  }
}
```

10.2 Client code — register for tokens

```
// lib/notifications.ts
import * as Notifications from 'expo-notifications';
import * as Device from 'expo-device';
import { Platform } from 'react-native';

export async function registerForPushAsync() {
  if (!Device.isDevice) return null;
  const { status: existingStatus } = await Notifications.getPermissionsAsync();
  let finalStatus = existingStatus;
  if (existingStatus !== 'granted') {
    const { status } = await Notifications.requestPermissionsAsync();
    finalStatus = status;
  }
  if (finalStatus !== 'granted') return null;
  const token = (await Notifications.getExpoPushTokenAsync()).data;
  return token;
}
```

In `AuthProvider`, after sign-in, call `registerForPushAsync()` and upload token to Supabase:

```
import { registerForPushAsync } from './notifications';
...
useEffect(() => {
  (async () => {
    if (user) {
      const token = await registerForPushAsync();
      if (token) {
        await supabase.from('profiles').update({ push_token: token }).eq('id',
user.id);
      }
    }
  })();
}, [user]);
```

10.3 Database changes

Add a `push_token` column to profiles:

```
alter table public.profiles add column if not exists push_token text;
```

10.4 Supabase Edge Functions — send notifications

Deploy two functions:

`notify-dm/index.ts`

```
import { serve } from 'https://deno.land/std@0.168.0/http/server.ts';

serve(async (req) => {
  const { record } = await req.json(); // new message
  const { sender_id, conversation_id, body } = record;

  const { data: members } = await supabase
    .from('conversation_members')
    .select('user_id, profiles(push_token, alias)')
    .eq('conversation_id', conversation_id);

  for (const m of members ?? []) {
    if (!m.profiles?.push_token || m.user_id !== sender_id) continue;
    await fetch('https://exp.host/--/api/v2/push/send', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
```

```

        to: m.profiles.push_token,
        title: 'New DM on CampusVoice',
        body: body.slice(0, 100)
      })
    });
  }
  return new Response('ok');
});

```

notify-post/index.ts (for new board posts)

```

serve(async (req) => {
  const { record } = await req.json(); // new post
  const { campus_id, body } = record;

  const { data: profiles } = await supabase
    .from('profiles')
    .select('push_token')
    .eq('campus_id', campus_id);

  for (const p of profiles ?? []) {
    if (!p.push_token) continue;
    await fetch('https://exp.host/--/api/v2/push/send', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        to: p.push_token,
        title: 'New post on your Campus Board',
        body: body.slice(0, 100)
      })
    });
  }
  return new Response('ok');
});

```

10.5 Database triggers

Link Edge Functions to table inserts:

```

create or replace function public.handle_new_message() returns trigger as $$
begin
  perform net.http_post(
    url := 'https://<your-project>.functions.supabase.co/notify-dm',
    headers := json_build_object('Content-Type', 'application/json'),

```

```

        body := row_to_json(NEW)
    );
    return NEW;
end;$$ language plpgsql;

create trigger on_new_message
after insert on public.messages
for each row execute function public.handle_new_message();

create or replace function public.handle_new_post() returns trigger as $$
begin
perform net.http_post(
    url := 'https://<your-project>.functions.supabase.co/notify-post',
    headers := json_build_object('Content-Type','application/json'),
    body := row_to_json(NEW)
);
return NEW;
end;$$ language plpgsql;

create trigger on_new_post
after insert on public.posts
for each row execute function public.handle_new_post();

```

Flow

- When a new **message** is inserted, Edge Function sends push to all other conversation members.
- When a new **post** is inserted, push goes to everyone in the same campus.

Now new DMs and campus board posts trigger **real-time push notifications** straight to Expo devices 

Update Pack — Push Notifications for new DMs & post replies

This adds **Expo push notifications** using an **Edge Function** so tokens stay private and anonymity is preserved. Flow: - App registers a device push token → stores it on the user's `profiles.expo_push_token`. - When you send a **DM** (insert into `messages`), the client pings the **Edge Function** with the `conversation_id`. The function (using the service key) finds the **other** member's token and sends a generic alert (no PII). - When someone **replies** to your post, the client pings the Edge Function with the `post_id`. The function looks up the post author's token and sends a notification.

1) Install client deps

```
npm i expo-notifications expo-device
```

2) SQL changes (tokens + replies)

Run this in Supabase SQL editor (after the earlier schema):

```
-- store Expo token per user (nullable)
alter table public.profiles add column if not exists expo_push_token text;

-- allow users to update only their own token
create policy if not exists "profiles_update_token_self"
  on public.profiles for update
  using (id = auth.uid())
  with check (id = auth.uid());

-- REPLIES table (post comments)
create table if not exists public.replies (
  id uuid primary key default gen_random_uuid(),
  post_id uuid not null references public.posts(id) on delete cascade,
  author_id uuid not null references auth.users(id) on delete cascade,
  body text not null check (length(body) <= 1500),
  created_at timestamptz not null default now()
);

alter table public.replies enable row level security;

-- Replies are campus-scoped implicitly via the post's campus; we only expose
reads via the post join on the client.
create policy if not exists "replies_insert_owner"
  on public.replies for insert
  with check (author_id = auth.uid());

create policy if not exists "replies_update_owner"
  on public.replies for update
  using (author_id = auth.uid());

create policy if not exists "replies_delete_owner"
  on public.replies for delete
  using (author_id = auth.uid());
```

3) Edge Function — notify

Create a Supabase **Edge Function** named `notify` (Deno). It uses the service role to look up tokens and calls Expo's push API.

supabase/functions/notify/index.ts

```
// deno-lint-ignore-file no-explicit-any
import { serve } from "https://deno.land/std@0.224.0/http/server.ts";

const SUPABASE_URL = Deno.env.get("SUPABASE_URL")!;
const SUPABASE_SERVICE_ROLE_KEY = Deno.env.get("SUPABASE_SERVICE_ROLE_KEY")!;
const EXPO_PUSH_URL = "https://exp.host/--/api/v2/push/send";

async function sb(path: string, init?: RequestInit) {
  const res = await fetch(`${SUPABASE_URL}/rest/v1/${path}`, {
    ...init,
    headers: {
      apikey: SUPABASE_SERVICE_ROLE_KEY,
      Authorization: `Bearer ${SUPABASE_SERVICE_ROLE_KEY}`,
      Prefer: "return=representation",
      ...init?.headers,
    },
  });
  if (!res.ok) throw new Error(await res.text());
  return res.json();
}

async function sendPush(to: string, title: string, body: string) {
  const res = await fetch(EXPO_PUSH_URL, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ to, title, body, sound: "default", priority:
"high" }),
  });
  if (!res.ok) throw new Error(await res.text());
}

type DMNotifyPayload = { conversation_id: string };
type ReplyNotifyPayload = { post_id: string };

await serve(async (req) => {
  try {
    const url = new URL(req.url);
    const type = url.searchParams.get("type");
    const payload = await req.json();
  }
}
```

```

    if (type === "dm") {
        const { conversation_id } = payload as DMNotifyPayload;
        // find both members
        const members = await sb(`conversation_members?
select=user_id&conversation_id=eq.${conversation_id}`) as any[];
        if (!members || members.length !== 2) return new Response("ok");

        // Who sent the request? (optional) If client includes x-user-id header,
        exclude that.
        const sender = req.headers.get("x-user-id");
        const recipientId = members.map(m => m.user_id).find((id: string) => id !== sender) ?? members[0].user_id;

        // lookup recipient token
        const profiles = await sb(`profiles?select=expo_push_token,alias&id=eq.${
recipientId}`) as any[];
        const token = profiles?.[0]?.expo_push_token;
        if (token) await sendPush(token, "New message", "You have a new anonymous
DM on CampusVoice.");
        return new Response("ok");
    }

    if (type === "reply") {
        const { post_id } = payload as ReplyNotifyPayload;
        // lookup post author
        const posts = await sb(`posts?select=author_id&id=eq.${post_id}`) as
any[];
        const authorId = posts?.[0]?.author_id;
        if (!authorId) return new Response("ok");
        const profiles = await sb(`profiles?select=expo_push_token&id=eq.${
authorId}`) as any[];
        const token = profiles?.[0]?.expo_push_token;
        if (token) await sendPush(token, "New reply", "Someone replied to your
campus post.");
        return new Response("ok");
    }

    return new Response("bad request", { status: 400 });
} catch (e) {
    return new Response(String(e?.message ?? e), { status: 500 });
}
});

```

Deploy

```
supabase functions deploy notify
supabase secrets set --project-ref YOUR_PROJECT_REF
  SUPABASE_URL=https://YOUR-PROJECT.supabase.co
  SUPABASE_SERVICE_ROLE_KEY=YOUR_SERVICE_ROLE
```

The function URL will be like:

```
https://YOUR-PROJECT.functions.supabase.co/notify
```

4) Client: register device token

Create `lib/notifications.ts` and call it after login/signup.

lib/notifications.ts

```
import * as Notifications from 'expo-notifications';
import * as Device from 'expo-device';
import { Platform } from 'react-native';
import { supabase } from '@/lib/supabase';

Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true,
    shouldPlaySound: true,
    shouldSetBadge: false,
  }),
});

export async function registerForPushAndSave() {
  if (!Device.isDevice) return;
  const { status: existingStatus } = await Notifications.getPermissionsAsync();
  let finalStatus = existingStatus;
  if (existingStatus !== 'granted') {
    const { status } = await Notifications.requestPermissionsAsync();
    finalStatus = status;
  }
  if (finalStatus !== 'granted') return;

  const token = (await Notifications.getExpoPushTokenAsync()).data;

  if (Platform.OS === 'android') {
    await Notifications.setNotificationChannelAsync('default', {
      name: 'default', importance: Notifications.AndroidImportance.MAX,
```



```

    });
  }

  // store token to your profile
  const { data: session } = await supabase.auth.getSession();
  const uid = session.session?.user?.id;
  if (!uid) return;
  await supabase.from('profiles').update({ expo_push_token: token }).eq('id', uid);
  return token;
}

```

lib/auth.tsx (add import and call after sign-in/signup):

```

import { supabase } from "@/lib/supabase";
+import { registerForPushAndSave } from "@/lib/notifications";
@@
  signIn: async (email, password) => {
    const { error } = await supabase.auth.signInWithPassword({ email,
password });
    if (error) throw error;
+    await registerForPushAndSave();
  },
  signUp: async (email, password) => {
@@
    if (error) throw error;
    const { error: initErr } = await supabase.rpc('init_profile');
    if (initErr) throw initErr;
+    await registerForPushAndSave();
  },

```

5) DM notifications — call Edge Function after sending

In **app/chat/[id].tsx**, after inserting the message, ping the function:

```

const send = async () => {
  if (!text.trim()) return;
  await supabase.from("messages").insert({ conversation_id: id, sender_id:
user?.id, body: text.trim() });
  setText("");
+  try {
+    const fnUrl = `${
{process.env.EXPO_PUBLIC_SUPABASE_URL!.replace('supabase.co', 'functions.supabase.co')}/
notify?type=dm`;

```

```

+      await fetch(fnUrl, {
+        method: 'POST',
+        headers: {
+          'Content-Type': 'application/json',
+          // optional: help the function identify sender to avoid self-
targeting
+          'x-user-id': user?.id ?? ''
+        },
+        body: JSON.stringify({ conversation_id: id })
+      });
+    } catch {}
  };

```

6) Post replies + notifications

SQL already added the `replies` table. Now update the Board UI to let students reply and notify the post author.

In `app/(tabs)/board.tsx`:

```

-      <View style={{ marginTop: 8 }}>
-        {item.author?.alias_code && (
-          <Pressable onPress={() => dm(item.author.alias_code)}>
-            <Text style={styles.dm}>DM this author →</Text>
-          </Pressable>
-        )}
-      </View>
+      <View style={{ marginTop: 8 }}>
+        {item.author?.alias_code && (
+          <Pressable onPress={() => dm(item.author.alias_code)}>
+            <Text style={styles.dm}>DM this author →</Text>
+          </Pressable>
+        )}
+      <ReplyBox postId={item.id} />
+    </View>

```

Add a tiny inline component at the bottom of the file:

```

function ReplyBox({ postId }: { postId: string }){
  const [val, setVal] = useState('');
  const send = async () => {
    if (!val.trim()) return;
    const { error } = await supabase.from('replies').insert({ post_id: postId,
author_id: (await supabase.auth.getUser()).data.user?.id, body: val.trim() });

```

```

    if (!error) {
      setVal('');
      try {
        const fnUrl = `${
process.env.EXPO_PUBLIC_SUPABASE_URL!.replace('supabase.co', 'functions.supabase.co')}/${
notify?type=reply`;
        await fetch(fnUrl, { method: 'POST', headers: { 'Content-Type':
'application/json' }, body: JSON.stringify({ post_id: postId }) });
      } catch {}
    }
  };
  return (
    <View style={{ flexDirection:'row', gap:8, marginTop:8 }}>
      <TextInput placeholder="Reply..." placeholderTextColor="#94a3b8"
value={val} onChangeText={setVal} style={{ flex:1, borderWidth:1,
borderColor:'#2b2f55', backgroundColor:'#0f1020', borderRadius:10,
color:'#fff', padding:10 }} />
      <Button label="Reply" onPress={send} />
    </View>
  );
}

```

7) Env note

No new env vars required on the client. Ensure your **Edge Function** has `SUPABASE_URL` and `SUPABASE_SERVICE_ROLE_KEY` secrets set.

Done!

You now have: - Push for **new DMs** - Push for **post replies** - All while keeping emails private and alias-only in the app.

Want a follow-up pack for **mute/block/report**, or for **batching** notifications (debounce DM spam)? I can wire either one next.