Perfect! Got it - I can see the inspiration clearly now. Love the clean, minimalist design with that light green/sage background!

Here's what I'll build for the **Discussion Forum** based on those images:

**Key Features I'll Include:**

1. **Discussion Topic Cards** - Each card will show:
   * Topic title (e.g., "Climate Change Adaptation Strategies")
   * Description/prompt for the discussion
   * Instructor name in green text (e.g., "Dr. Lisa Wong")
   * Institution/Academy name in gray
   * Stats: reply count, participant count, time posted
   * "Join Discussion" button (green outlined)
2. **Discussion Thread View** - When you click into a topic:
   * Full topic details at top
   * List of essay responses from students
   * Reply/Write Essay button
   * Each essay showing author, timestamp, content
3. **Essay Submission Form**:
   * Text area for writing essays
   * Character/word count
   * Submit button
4. **Style Matching**:
   * Light sage/mint green background
   * Green accents for text and buttons
   * Clean card-based layout
   * Icons for replies, participants, time
   * Soft shadows on cards

**Before I Build:**

Yes! Please share your **homepage HTML/CSS** so I can match:

* Exact color codes
* Font families
* Button styles
* Header/navigation style
* Any existing CSS variables or classes

Once you share that, I'll create a perfectly themed discussion forum that fits seamlessly with your EcoLearn site! 🌿

Retry

<!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8" /> <meta name="viewport" content="width=device-width, initial-scale=1.0" /> <title>EcoLearn</title> <link rel="stylesheet" href="test.css" /> <link rel="stylesheet" href="https://cdnjs.cloudflare

pasted

@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=swap'); @import url('https://fonts.googleapis.com/css2?family=Jersey+10&family=Jersey+15&family=Pixelify+Sans:wght@400..700&family=Ubuntu:wght@300;400;500;700&display=swap'); \* { margin: 0; padding

pasted

F

yes the theme is not of the pics im sharng the html and css for the theme there u go

Perfect! I've got your theme now! Love the **#B2E0E6 (light cyan/turquoise)** and **#5EAAB5 (teal)** color scheme with the Poppins font. Clean and modern! 🌊

Now I'll build the **Discussion Forum** that matches your EcoLearn theme perfectly. Here's what I'm creating:

**Discussion Forum Features:**

1. **Forum List View** (Main page):
   * Topic cards with clean design matching your style
   * Shows: title, description, instructor name (green), institution, stats (replies, participants, time)
   * "Join Discussion" button styled like your buttons
   * Tab navigation between "Student Posts" and "Discussion Forum"
2. **Discussion Thread View** (Individual topic):
   * Full topic details at top
   * List of essay submissions
   * "Write Essay" button to add responses
   * Back button to return to forum list
3. **Essay Submission Modal**:
   * Pop-up form for writing essays
   * Text area with word count
   * Submit button
   * Same modal styling as your login modals
4. **Styling**:
   * Uses your exact colors: #B2E0E6, #5EAAB5, #C4F1F4
   * Poppins font family
   * Same button styles and hover effects
   * Matching border radius and shadows

Ready to build it now! I'll create a single HTML file with embedded CSS and JavaScript. Should I proceed? 🚀

erfect! I've created **3 separate files** for you:

**✅ Files Created:**

1. **discussion-forum.html** - Complete HTML structure
2. **discussion-forum.css** - All styling matching your EcoLearn theme perfectly
3. **discussion-forum.js** - Full JavaScript functionality

**🎨 Theme Features Used:**

* **Colors**: #B2E0E6, #5EAAB5, #C4F1F4 (your exact colors!)
* **Font**: Poppins (matching your site)
* **Buttons**: Same border-radius (30px) and hover effects
* **Cards**: Same shadow and styling as your feature cards

**🚀 Current Features:**

* ✅ Discussion topic cards with instructor info
* ✅ Click to open full discussion thread
* ✅ View all essays/responses
* ✅ Write and submit new essays
* ✅ Word counter in essay form
* ✅ Modal popup (styled like your login modal)
* ✅ Fully responsive design

**📌 Next Steps:**

Now we need to connect this to **Supabase**! We'll need to:

1. Create tables in Supabase for discussions and essays
2. Replace the sample data with Supabase queries
3. Add real-time data fetching and submission

**Ready to integrate Supabase?** Let me know and I'll help you set it up! 🌱

few changes we need to do is that is aesthetically same blue but softer and remove the fgreen ake the post cards maybe frosted with some color background and the text more better and the font better when the user submits the essay the user name should also be there liek userid when we submit the essay alos instead of name we need name nad user id as the parameters also make it more aesthetic but keep it in theme it should not be different when u opne discussion forum also remove nav bar on main page its the otehr teams job to out to gether different pages if u want for aesthetic i will give the html and css for sign up page so u get idea ??

Perfect! I get what you're going for - **softer blues, remove green, frosted glass cards, better fonts, and include userID for essays**.

Yes, please **share the signup page HTML and CSS** so I can match that aesthetic exactly! That'll help me understand:

* The frosted glass effect you want
* Better font styling
* Color palette refinement
* Overall vibe

Once I see that, I'll update all 3 files to:

* ✅ Remove navbar (discussion forum will be standalone)
* ✅ Softer blue tones throughout
* ✅ Remove all green accents
* ✅ Frosted glass cards with colored backgrounds
* ✅ Better typography and text hierarchy
* ✅ Add both **Name** and **UserID** fields in essay submission
* ✅ Display userID alongside author name in essays
* ✅ Make the thread view more aesthetic
* ✅ Keep everything cohesive with your theme

Go ahead and share the signup page files! 🎨

---------------------------------------------------------------------------------------------------------------   
  
Nice — using the **same Supabase project** for both Student Posts and Discussion Forum is the right call. I’ll walk you step-by-step from DB design → auth → frontend wiring → file avatars → realtime updates and security rules. I’ll give ready-to-drop code snippets you can paste into your project (supabase.js, updated discussion-forum.js snippets, SQL for tables & RLS policies). Let’s go.

# 1) Quick plan (what we’ll do)

1. Create DB tables: discussions, essays.
2. Add storage bucket for avatars.
3. Configure Auth (email / Google, etc.).
4. Add Row Level Security (RLS) policies.
5. Add supabase.js client to the frontend.
6. Replace local discussions array with real DB reads/writes.
7. Upload avatars to storage and show them in essays.
8. Optionally: add realtime subscriptions so new essays appear automatically.

# 2) Use the same Supabase project?

Yes — use the same project. Put **all** EcoLearn tables and storage in that one project. Invite teammates as collaborators via Supabase Dashboard. No need for multiple Gmail accounts; invite them by email.

# 3) Database schema (SQL)

Open Supabase → SQL Editor → run these queries to create tables.

-- discussions table (teacher creates topics)

create table public.discussions (

id bigint generated by default as identity primary key,

title text not null,

description text,

instructor text,

institution text,

replies int default 0,

participants int default 0,

created\_at timestamptz default now()

);

-- essays table (student replies)

create table public.essays (

id bigint generated by default as identity primary key,

discussion\_id bigint not null references public.discussions(id) on delete cascade,

user\_id uuid, -- will reference auth.users.id (not enforced in foreign key because auth schema)

author\_name text not null,

user\_identifier text, -- your display user id (like STU2024xxx)

content text not null,

created\_at timestamptz default now()

);

-- index for performance

create index on public.essays (discussion\_id);

create index on public.essays (created\_at);

# 4) Storage: create bucket for avatars

1. In Supabase dashboard → Storage → New bucket (name it avatars)
2. Make it **private** (recommended) so only signed-in users can access via signed URLs. You can make public if you prefer simpler URLs.

# 5) Enable Auth providers

In Supabase → Authentication → Settings:

* Enable Email/Password (default).
* Optional: Google (requires OAuth client ID).

We will use supabase.auth on the frontend.

# 6) Row Level Security (RLS) — secure your data

Turn on RLS for tables (Supabase → Table Editor → select table → Policies → enable RLS). Then add policies:

**For discussions**

* Allow insert by teacher role (we’ll show how to mark teacher in users metadata).
* select allowed for public (so everyone can read).

Example SQL policies:

-- allow everyone to select discussions

create policy "public\_select" on public.discussions

for select using (true);

-- allow only teachers to insert discussions (assumes role 'teacher' in auth.users app\_metadata)

create policy "teachers\_insert" on public.discussions

for insert using (auth.role() = 'authenticated' and (auth.jwt() ->> 'role') = 'teacher');

**For essays**

* Allow insert by authenticated users.
* select allowed for public.
* Allow delete / update only by owner or teacher.

Example policies:

-- allow anyone to read essays

create policy "essays\_select\_public" on public.essays

for select using (true);

-- allow authenticated users to insert essays (we will set user\_id to auth.uid())

create policy "essays\_insert\_auth" on public.essays

for insert with check (auth.role() = 'authenticated');

-- allow users to update/delete their own essays

create policy "essays\_update\_owner" on public.essays

for update using (user\_id = auth.uid());

create policy "essays\_delete\_owner" on public.essays

for delete using (user\_id = auth.uid());

-- allow teacher to delete/update any essay (similar to insert)

create policy "essays\_update\_teacher" on public.essays

for update using ((auth.jwt() ->> 'role') = 'teacher');

create policy "essays\_delete\_teacher" on public.essays

for delete using ((auth.jwt() ->> 'role') = 'teacher');

**Note:** Supabase’s auth.jwt() returns claims from the JWT. You’ll need to set a role custom claim or store teacher flag in a separate profile table and check it. If you don’t want JWT claim complexity yet, use a profiles table and check membership there in policies. I can provide the profiles flow if you want.

# 7) Create a small profiles table (recommended)

This stores avatar URL, display name, role:

create table public.profiles (

id uuid primary key references auth.users (id) on delete cascade,

full\_name text,

avatar\_url text,

role text default 'student',

created\_at timestamptz default now()

);

Policy example: let users insert/update their own profile:

create policy "profiles\_select\_public" on public.profiles for select using (true);

create policy "profiles\_upsert\_self" on public.profiles for insert with check (auth.uid() = id);

create policy "profiles\_update\_self" on public.profiles for update using (auth.uid() = id);

# 8) Frontend: supabase.js (single client file)

Create supabase.js in your project root, export the client.

// supabase.js

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

export const SUPABASE\_URL = 'https://YOUR\_PROJECT\_REF.supabase.co';

export const SUPABASE\_ANON\_KEY = 'YOUR\_ANON\_PUBLIC\_KEY';

export const supabase = createClient(SUPABASE\_URL, SUPABASE\_ANON\_KEY);

**Where to find keys:** Supabase Dashboard → Settings → API → Project URL + anon key.  
**Security note:** The anon key is safe for client use when RLS is in place. Don’t push service\_role key to the client.

# 9) Replace local data with DB reads (key functions)

Below are the key JS functions to add into your discussion-forum.js to use Supabase. I’ll show fetchDiscussions, fetchDiscussionById (essays included), submitEssay, plus avatar logic.

Add at top of your discussion-forum.js:

import { supabase } from './supabase.js';

Then use these functions (examples):

// fetch all discussions

async function fetchDiscussionsFromDb() {

const { data, error } = await supabase

.from('discussions')

.select('\*')

.order('created\_at', { ascending: false });

if (error) return console.error('fetch discussions err', error);

return data;

}

// fetch discussion and its essays

async function fetchDiscussionWithEssays(discussionId) {

// We'll fetch discussion and essays separately (or use RPC/joins)

const { data: discussion } = await supabase

.from('discussions')

.select('\*')

.eq('id', discussionId)

.single();

const { data: essays } = await supabase

.from('essays')

.select('\*, profiles:profiles(full\_name, avatar\_url)') // optional join if using FK to profiles

.eq('discussion\_id', discussionId)

.order('created\_at', { ascending: false });

return { discussion, essays };

}

// submit new essay

async function submitEssayToDb({ discussionId, content, userId, authorName, userIdentifier }) {

// make sure user is authenticated; supabase will set auth.uid() if you set user\_id server-side via trigger or use client to provide it

const { data, error } = await supabase

.from('essays')

.insert([{

discussion\_id: discussionId,

user\_id: userId, // from supabase.auth.getUser()

author\_name: authorName,

user\_identifier: userIdentifier,

content: content

}])

.select()

.single();

if (error) throw error;

return data;

}

# 10) Wire these into your UI (simplified)

Replace your initial renderDiscussions() init with DB fetch:

async function initForum() {

const rows = await fetchDiscussionsFromDb();

// map DB rows into the discussion-card UI

if (rows) {

discussionsContainer.innerHTML = '';

rows.forEach(d => {

// convert to expected shape or modify createDiscussionCard to accept DB row directly

discussionsContainer.appendChild(createDiscussionCardFromRow(d));

});

}

attachEventListeners();

}

When opening a discussion, fetch essays:

async function openDiscussion(discussionId) {

currentDiscussionId = discussionId;

const { discussion, essays } = await fetchDiscussionWithEssays(discussionId);

// render threadHeader from discussion

// convert essays DB rows to the shape expected by createEssayCard (including avatar)

renderEssaysFromDbRows(essays);

forumListView.style.display = 'none';

threadView.style.display = 'block';

}

# 11) Auth — get current user & link user\_id

Add basic auth functions (sign-in/out) or use Supabase UI later. For now, just get current user:

// get user

const { data: { user } } = await supabase.auth.getUser();

// or

const user = supabase.auth.getUser().then(r => r.data.user);

// when inserting essay use user.id as user\_id

await submitEssayToDb({

discussionId: currentDiscussionId,

content,

userId: user.id,

authorName: user.user\_metadata.full\_name || 'Anonymous',

userIdentifier: /\* your student id input or from profile \*/

});

**Note:** When using supabase.auth.onAuthStateChange you can react to logins and prefill author fields automatically.

# 12) Avatars: upload and display

**Uploading (on profile page):**

async function uploadAvatar(file, userId) {

const fileExt = file.name.split('.').pop();

const fileName = `${userId}.${fileExt}`;

const { data, error } = await supabase.storage

.from('avatars')

.upload(fileName, file, { upsert: true });

if (error) throw error;

// get public URL (if bucket public) or signed URL for private

const { data: publicUrlData } = supabase.storage.from('avatars').getPublicUrl(fileName);

return publicUrlData.publicUrl; // or fetchSignedUrl if private

}

**Displaying avatar in essay card**

* If profiles.avatar\_url exists, build <img src="..."> inside .essay-avatar.
* If not, show the Font Awesome fallback.

Example in createEssayCard:

const avatarHtml = e.profiles?.avatar\_url

? `<img src="${e.profiles.avatar\_url}" alt="${e.author}" style="width:48px;height:48px;border-radius:50%;">`

: `<i class="fas fa-user"></i>`;

card.innerHTML = `

<div class="essay-header">

<div class="essay-author-info">

<div class="essay-avatar">${avatarHtml}</div>

<div> ... </div>

</div>

...

</div>

...

`;

If bucket is private, use signed URLs:

const { data } = await supabase.storage.from('avatars').createSignedUrl(fileName, 60); // 60 seconds

const url = data.signedUrl;

# 13) Realtime (optional)

To get new essays in real time:

supabase.channel('public:essays')

.on('postgres\_changes', { event: 'INSERT', schema: 'public', table: 'essays' }, payload => {

// payload.new contains inserted row

if (payload.new.discussion\_id === currentDiscussionId) {

// push to UI

prependEssayToUI(payload.new);

}

})

.subscribe();

# 14) Putting it together — minimal flow for Submit Essay

1. User logs in → supabase.auth.getUser() returns user.id.
2. User opens discussion → openDiscussion() fetches essays.
3. User enters essay and clicks submit → your submitEssay() gets user info and calls submitEssayToDb() above.
4. DB returns row → frontend renderEssays() updates UI.

# 15) Important security notes

* NEVER use the **service\_role** key in client code. Use anon key + RLS.
* Always enable RLS and write policies before exposing anon key to the web.
* Test policies with an unprivileged user account to confirm behavior.

# 16) Example: full submitEssay() hooked to your form (client-side JS)

Drop this into your discussion-forum.js replacing the local insert logic:

async function submitEssay(e) {

e.preventDefault();

const content = essayContent.value.trim();

const userIdentifier = userIdInput.value.trim();

const authorName = authorInput.value.trim();

if (!currentDiscussionId) return alert('Open a discussion first.');

if (!authorName || !userIdentifier || !content) return alert('Please fill all fields.');

// ensure user is logged in

const { data: { user } } = await supabase.auth.getUser();

if (!user) {

return alert('Please sign in to submit an essay.');

}

try {

const inserted = await submitEssayToDb({

discussionId: currentDiscussionId,

content,

userId: user.id,

authorName,

userIdentifier

});

// optionally: update in-memory data and UI

renderEssays(await getEssaysForDiscussion(currentDiscussionId));

closeEssayModal();

} catch (err) {

console.error(err);

alert('Failed to submit. Try again.');

}

}

getEssaysForDiscussion should call Supabase to fetch essays for that discussion.

# 17) Testing checklist (do this in order)

1. Create tables + profiles + storage bucket + RLS policies.
2. Add supabase.js to project with project URL + anon key.
3. Implement initForum() to fetch discussions from DB.
4. Implement openDiscussion() to fetch essays for that discussion.
5. Implement auth sign-up/sign-in (at least email) or use Supabase UI.
6. Submit essay as logged-in user — verify DB row inserted and UI updates.
7. Upload profile avatar — verify it appears in essay card.
8. Test policies by trying to submit while logged out (should fail).

# 18) Want me to implement the code into your repo?

I can paste a full updated discussion-forum.js based on your existing UI that:

* Imports supabase.js,
* Replaces the local discussions array with real DB fetch,
* Implements submitEssayToDb, avatar rendering, and realtime subscription.

Tell me:

* Do you want avatars stored as **public** or **private** bucket? (public = simpler URLs; private = signed URLs & safer)
* Do you already have a profiles table or do you want me to include the creation SQL above?

If you say “yes, implement”, I’ll generate the full client JS ready to copy into your project.

--------------------------------------------------------------------------------------------------------------------------------------------------  
  
  
  
Perfect — that’s totally fine, Fawaz 👍

Since you haven’t built authentication yet, we’ll just:  
✅ Use a **temporary name and user ID input** (from your current modal form).  
✅ Store essays and discussions directly into Supabase under those fields.  
✅ Later, once you add authentication, we’ll **replace those manual inputs** with supabase.auth.getUser() automatically.

Here’s the exact next step plan for you 👇

### 🧭 PHASE 1 — Connect Your Existing Supabase Project

1. Go to your Supabase dashboard → open your **EcoLearn** project.
2. Click **SQL Editor → New Query**, and paste this:

CREATE TABLE public.discussion\_threads (

id uuid DEFAULT gen\_random\_uuid() PRIMARY KEY,

title text NOT NULL,

description text,

created\_by text,

created\_at timestamptz DEFAULT now()

);

CREATE TABLE public.discussion\_essays (

id uuid DEFAULT gen\_random\_uuid() PRIMARY KEY,

thread\_id uuid REFERENCES discussion\_threads (id) ON DELETE CASCADE,

author\_name text NOT NULL,

user\_id text NOT NULL,

content text NOT NULL,

created\_at timestamptz DEFAULT now()

);

→ Hit **Run** ✅

That creates the two discussion tables inside your same project.

### 🧭 PHASE 2 — Connect Supabase in Your discussion-forum.js

At the **very top** of your discussion-forum.js file, add this:

import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

const SUPABASE\_URL = "https://YOUR\_PROJECT\_ID.supabase.co";

const SUPABASE\_ANON\_KEY = "YOUR\_ANON\_KEY";

const supabase = createClient(SUPABASE\_URL, SUPABASE\_ANON\_KEY);

You’ll find those two values under:  
**Project Settings → API → Project URL** and **anon public key**

### 🧭 PHASE 3 — Replace “Load Discussion” section with Supabase fetch

Replace your current loadDiscussions() or the part that loads your forum cards with this:

async function loadDiscussions() {

const { data, error } = await supabase

.from("discussion\_threads")

.select("\*")

.order("created\_at", { ascending: false });

if (error) {

console.error("Error loading discussions:", error);

return;

}

const container = document.getElementById("discussionsContainer");

container.innerHTML = "";

data.forEach(thread => {

const div = document.createElement("div");

div.className = "discussion-card";

div.innerHTML = `

<h3>${thread.title}</h3>

<p>${thread.description}</p>

<button onclick="openThread('${thread.id}')">Join Discussion</button>

`;

container.appendChild(div);

});

}

document.addEventListener("DOMContentLoaded", loadDiscussions);

### 🧭 PHASE 4 — Add openThread() and loadEssays() functions

Below that, add:

let currentThreadId = null;

async function openThread(threadId) {

currentThreadId = threadId;

document.getElementById("forum-list-view").style.display = "none";

document.getElementById("thread-view").style.display = "block";

await loadEssays(threadId);

}

async function loadEssays(threadId) {

const { data: essays, error } = await supabase

.from("discussion\_essays")

.select("\*")

.eq("thread\_id", threadId)

.order("created\_at", { ascending: false });

if (error) {

console.error("Error loading essays:", error);

return;

}

const container = document.getElementById("essaysContainer");

container.innerHTML = "";

essays.forEach(essay => {

const div = document.createElement("div");

div.className = "essay-card";

div.innerHTML = `

<div class="essay-header">

<i class="fas fa-user-circle"></i>

<h4>${essay.author\_name}</h4>

</div>

<p>${essay.content}</p>

`;

container.appendChild(div);

});

}

### 🧭 PHASE 5 — Handle Essay Submission

At the bottom of the file:

document.getElementById("essayForm").addEventListener("submit", async (e) => {

e.preventDefault();

const authorName = document.getElementById("authorName").value.trim();

const userId = document.getElementById("userId").value.trim();

const essayContent = document.getElementById("essayContent").value.trim();

if (!authorName || !userId || !essayContent) {

alert("Please fill all fields before submitting.");

return;

}

const { error } = await supabase

.from("discussion\_essays")

.insert({

thread\_id: currentThreadId,

author\_name: authorName,

user\_id: userId,

content: essayContent

});

if (error) {

alert("Error posting essay: " + error.message);

} else {

closeEssayModal();

loadEssays(currentThreadId);

}

});

### 🧭 PHASE 6 — Test It

1. Open your discussion-forum.html in your browser (from Live Server or localhost).
2. Check the console (F12 → Console tab).
3. See if it logs any “error loading” messages.
4. If you don’t have any threads yet, manually insert one into Supabase:
   * Go to your table → discussion\_threads → **Insert Row**
   * Example:
     + title: “How to reduce plastic waste?”
     + description: “Share your practical solutions for plastic pollution.”
     + created\_by: “Teacher Admin”

Reload your page → it’ll appear 🎉

Would you like me to now build the **complete JS file** with all these steps already coded and formatted — ready to replace your old discussion-forum.js?  
(That way you can just plug it in and it’ll instantly work with Supabase.)

Here’s what I’ll do when I modify it:

### ✅ Phase 3 – Fetch Discussions from Supabase

* Replace your hard-coded discussions array with data fetched from a new table discussions in Supabase.
* Each discussion record will have fields like id, title, description, instructor, institution, replies, participants, time\_ago.

### ✅ Phase 4 – Fetch Essays for Each Discussion

* When you open a discussion, it will load essays from a related table essays.
* The essays table will store: id, discussion\_id, author, user\_id, content, created\_at, and optionally pfp\_url.

### ✅ Phase 5 – Submit Essays to Supabase

* When someone submits an essay, it will insert the new essay into the essays table and then instantly show it in the UI (no refresh).

Before I send your full upgraded JS, you’ll need **two new tables** in Supabase:

#### 🧱 Table 1: discussions

| **column** | **type** | **description** |
| --- | --- | --- |
| id | uuid | Primary key (default gen\_random\_uuid()) |
| title | text | Discussion topic title |
| description | text | Description or prompt |
| instructor | text | Instructor name |
| institution | text | School/Institute |
| replies | integer | Number of essays (optional) |
| participants | integer | Unique contributors (optional) |
| time\_ago | text | “2 hours ago”, etc. |

#### 🧱 Table 2: essays

| **column** | **type** | **description** |
| --- | --- | --- |
| id | uuid | Primary key (default gen\_random\_uuid()) |
| discussion\_id | uuid | Foreign key → discussions.id |
| author | text | Essay author name |
| user\_id | text | Student ID |
| content | text | Essay content |
| pfp\_url | text | Profile picture URL (optional) |
| created\_at | timestamptz | Default now() |

Once you **create these tables**, send me a quick ✅ confirmation (and optionally what table names you used if you changed them),  
and I’ll return your **final JS file** with:

* Full Supabase CRUD (fetch + post essays)
* Local fallback if offline
* Smooth animations and modal UX intact
* PFP support (shows default icon until authentication is added)

Ready?