perplexity

Lets add Authentication and Authorization to it using Supabase.

Integrating Supabase for authentication and authorization in your Astro-powered TETRIX site provides secure user management, role-based access control, and seamless session handling. Below is a step-by-step guide—complete with code snippets and UI/UX notes—to equip your frontend developer with the tasks and stories needed to implement signup, login, protected routes, profile management, and RBAC.

Supabase Setup and Configuration

- Create a Supabase project and enable Email/Password (and any OAuth) providers in the Supabase dashboard under **Authentication > Providers**[1].
- Add your Supabase URL and anon key to a `.env` file at your project root:

  ```
  PUBLIC_SUPABASE_URL=https://your-project.supabase.co
  PUBLIC_SUPABASE_ANON_KEY=your-anon-key
  ```

  Then declare them in `src/env.d.ts` for IntelliSense support[1].
- Install dependencies:

  ```
  npm install @supabase/supabase-js supabase-auth-helpers-astro
  ```

  `supabase-auth-helpers-astro` provides Astro-optimized utilities for client and server auth[2].

Initialize the Supabase Client

Create `src/lib/supabase.ts` to instantiate Supabase across the app[1] [2]:

```
import { createClient } from '@supabase/supabase-js';

export const supabase = createClient(
  import.meta.env.PUBLIC_SUPABASE_URL,
  import.meta.env.PUBLIC_SUPABASE_ANON_KEY
);
```

Authentication UI Components

User Stories:

- As a visitor, I can sign up with email/password.
- As a user, I can log in, log out, and see my profile.

Implementation:

1. In `src/components/auth/AuthForm.tsx` (client component), use the `use client` directive and Tailwind for styling:

```
'use client';
import { useState } from 'react';
import { supabase } from '@/lib/supabase';

export default function AuthForm() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const handleSignup = async () => {
    await supabase.auth.signUp({ email, password });
  };
  const handleLogin = async () => {
    await supabase.auth.signInWithPassword({ email, password });
  };
  return (
    <div className="max-w-md mx-auto p-6 bg-white shadow rounded">
      <input
        className="w-full mb-4 p-2 border rounded"
        type="email"
        placeholder="Email"
        value={email}
        onChange={e => setEmail(e.target.value)}
      />
      <input
        className="w-full mb-4 p-2 border rounded"
        type="password"
        placeholder="Password"
        value={password}
        onChange={e => setPassword(e.target.value)}
      />
      <button
        className="w-full mb-2 py-2 bg-gray-800 text-white rounded"
        onClick={handleLogin}
      >
        Log In
      </button>
      <button
        className="w-full py-2 bg-gray-200 text-gray-800 rounded"
        onClick={handleSignup}
      >
        Sign Up
      </button>
    </div>
  );
}
```

2. Render `<AuthForm />` in `src/pages/login.astro` or within a modal.

Protecting Routes in Astro

Use Astro middleware to redirect unauthorized users[3]:

```
// src/middleware.ts
import type { MiddlewareHandler } from 'astro';
import { supabase } from './lib/supabase';

export const onRequest: MiddlewareHandler = async ({ request, redirect }) => {
  const { data: { session } } = await supabase.auth.getSession({
    cookieOptions: { request },
  });
  if (!session && request.url.pathname.startsWith('/dashboard')) {
    return redirect('/login');
  }
};
```

Pages under `/dashboard` will now require authentication.

Authorization with RLS and RBAC

- **Row Level Security:** Enable RLS on tables in the Supabase dashboard or via SQL[4].
- **Policies:** Create policies per table. Example for profiles:

  ```
  alter table public.profiles enable row level security;
  create policy "Users can view their own profile"
    on public.profiles for select
    using (auth.uid() = id);
  ```

- **Role-Based Access Control:** Define roles in Supabase under **Authentication > Policies** or using a `roles` table[5]. Assign roles (e.g. `admin`, `user`) and write RLS policies accordingly.

User Management Stories and Tasks

- **Signup Flow:** Validate input, call `supabase.auth.signUp`, handle email confirmation.
- **Login Flow:** Call `supabase.auth.signInWithPassword`, store session in cookies.
- **Logout:** Call `supabase.auth.signOut` and clear cookies.
- **Profile Page:** Fetch user metadata from `auth.user()` and display/edit profile fields; call `supabase.auth.updateUser` for changes.
- **Protected Dashboard:** Only users with `role = 'admin'` can access `/dashboard/admin`, enforced via additional RLS policies.

Best Practices

- Store session cookies **httpOnly** and **secure**.
- Use `supabase.auth.onAuthStateChange` in client components to reactively update UI.
- Prefetch user session on the server using Astro's server–client split for faster page loads.
- Thoroughly test RLS policies to avoid blocking legitimate queries[6].

- Regularly rotate anon keys and revoke inactive sessions.

Adding Supabase authentication and authorization layers ensures TETRIX users can securely register, log in, and access resources according to their roles—all within Astro's islands architecture.

⁂

1. https://docs.astro.build/en/guides/backend/supabase/
2. https://github.com/AceCodePt/supabase-auth-helpers-astro
3. https://mihai-andrei.com/blog/how-to-add-supabase-auth-to-astro/
4. https://supabase.com/docs/guides/database/postgres/row-level-security
5. https://supabase.com/features/role-based-access-control
6. https://supabase.com/docs/guides/auth/managing-user-data