Guía completa (Canvas) — Echo(s) con Angular 20

Objetivo: finalizar el *frontend* del proyecto **Echo(s)** usando **Angular 20** moderno: *standalone components*, `provideRouter`, **control flow** (`@if`, `@for`, `@empty`), **guards funcionales**, **signals** y **servicios**. La guía es **lineal**, **minimalista** y usa **un solo environment** para evitar ruido.

> Filosofía: separar **presentación (shared)** de **lógica de datos (core/services)** y **pantallas (features)**. Entregar fragmentos (no todo el código) para fomentar el aprendizaje.

---

0) Requisitos

- Node 18+ y Angular CLI (`npm i -g @angular/cli`)
- Proyecto Supabase listo (URL + anon key)

---

1) Estructura simple y coherente

```
ng new echo-app --style=scss --routing=false
cd echo-app
npm i @supabase/supabase-js
```

Árbol de carpetas (propuesto):

```
src/
  app/
    app.config.ts
    app.routes.ts

    core/                 # motor de la app (datos, auth, tipos)
      environment/
        environment.ts
      supabase/
        supabase.service.ts
      auth/
        auth.service.ts
        auth.guard.ts
        guest.guard.ts
      models/
        echo.model.ts
        profile.model.ts
        like.model.ts
        comment.model.ts
      services/
```

```
        echos.service.ts
        profiles.service.ts
        likes.service.ts
        comments.service.ts

    shared/               # UI presentacional (sin llamadas a API)
      button/
        button.component.ts
      echo-card/
        echo-card.component.ts

    features/             # páginas (una carpeta por ruta)
      auth/
        login.page.ts
        signup.page.ts
      feed/
        feed.page.ts
      echo-detail/
        echo-detail.page.ts
      profile/
        profile.page.ts
      liked/
        liked.page.ts

  styles.scss
  main.ts
```

Reglas:

- **core** = servicios y tipos reutilizables.
- **shared** = componentes visuales reusables (sin conocer servicios).
- **features** = páginas; orquestan servicios y componen la UI.

Checklist: crea **todos los archivos vacíos** primero y ejecuta `ng serve` para detectar errores temprano (*fail fast*).

---

2) Un solo `environment`

Archivo: `src/app/core/environment/environment.ts`

```ts
// fragmento
export const environment = {
  appName: 'Echo App (Edu)',
  supabaseUrl: 'https://TU-PROYECTO.supabase.co',
  supabaseAnonKey: 'TU_ANON_KEY',
  pageSize: 10,
  enableRealtime: true, // opcional para suscripciones
};
```

Aquí vive todo lo configurable. **No** hay replacements ni archivos por ambiente.

Prueba mental: si mañana cambias `pageSize`, ¿solo tocas este archivo? ✔️

---

3) Cliente de Supabase (servicio base)

Archivo: `src/app/core/supabase/supabase.service.ts`

Idea: un único servicio que crea y expone `SupabaseClient`. Sin lógica de negocio.

```
// fragmento
@Injectable({ providedIn: 'root' })
export class SupabaseService {
  client = createClient(environment.supabaseUrl,
environment.supabaseAnonKey);
}
```

Smoke test: inyecta en una página y `console.log(this.supa.client)`.

---

4) Modelos (tipos de datos, sin lógica)

Archivo: `src/app/core/models/echo.model.ts`

```
// fragmento mínimo
export interface Echo {
  id: number;
  content: string;
  image_url?: string | null;
  created_at: string;         // ISO
  user_id: string;            // UUID
  profile?: { username: string; avatar_url?: string | null };
  likes_count?: number;
  comments_count?: number;
}
```

Crea también:

- `profile.model.ts` → `{ id, username, avatar_url?, updated_at? }`
- `like.model.ts` → `{ id, echo_id, user_id, created_at }`
- `comment.model.ts` → `{ id, echo_id, user_id, content, created_at }`

Tip: los modelos **no** importan servicios.

---

5) Componentes UI (shared) — presentacionales

5.1) Button

Archivo: `src/app/shared/button/button.component.ts`

```
// fragmento
@Component({
  standalone: true,
  selector: 'ui-button',
  template: `<button class="btn" [disabled]="disabled"><ng-content /></
button>`,
})
export class ButtonComponent { @Input() disabled = false; }
```

- Estila `.btn` en `styles.scss` (focus accesible, tamaño táctil ≥ 44×44).
- No debe conocer servicios.

5.2) EchoCard

Archivo: `src/app/shared/echo-card/echo-card.component.ts`

```
// fragmento
@Component({
  standalone: true,
  selector: 'ui-echo-card',
  template: `
    <article class="card">
      <header>
        <img
*ngIf="echo.profile?.avatar_url" [src]="echo.profile?.avatar_url"
class="avatar" alt="" />
        <strong>@{{ echo.profile?.username }}</strong>
        <time class="muted">{{ echo.created_at | date:'short' }}</time>
      </header>

      <p>{{ echo.content }}</p>
      <img *ngIf="echo.image_url" [src]="echo.image_url" class="w-full"
alt="" />

      <footer class="row">
        <ui-button (click)="onLike?.()">♀ Like</ui-button>
        <ui-button (click)="onUnlike?.()"># Quitar</ui-button>
      </footer>
    </article>
  `,
  imports: [/* ButtonComponent, CommonModule si lo usas */],
})
export class EchoCardComponent {
```

```
  @Input() echo!: Echo;
  @Input() onLike?: () => void;
  @Input() onUnlike?: () => void;
}
```

- Recibe datos/callbacks. **Nunca** llama a Supabase ni navega por sí misma.

---

6) Servicios de dominio (core/services)

Servicios pequeños y enfocados.

6.1) AuthService (estado + métodos)

Archivo: `src/app/core/auth/auth.service.ts`

Responsabilidades:

- Mantener sesión (ideal con **signals**).
- `initSession()`, `signInWithPassword()`, `signUp()`, `signOut()`.
- `isAuthenticated()` para guards.

```
// fragmento
@Injectable({ providedIn: 'root' })
export class AuthService {
  private sessionSig = signal<any | null>(null);
  session = this.sessionSig.asReadonly();
  constructor(private supa: SupabaseService) {}
  async initSession(){ /* getSession + onAuthStateChange -> sessionSig.set
*/ }
  isAuthenticated(){ return !!this.session(); }
}
```

6.2) EchosService

Archivo: `src/app/core/services/echos.service.ts`

Responsabilidades:

- Listado paginado, detalle con conteos, crear echo.

```
// fragmento (paginación)
getPage(page: number) {
  const from = page * environment.pageSize, to = from + environment.pageSize
- 1;
  return this.supa.client
    .from('echos')
    .select('*, profiles(username, avatar_url)')
```

```
    .order('created_at', { ascending: false })
    .range(from, to);
}
```

6.3) ProfilesService / LikesService / CommentsService

- `profiles.service.ts`: `getProfile(id)`, `updateProfile(partial)`.
- `likes.service.ts`: `like(echoId, userId)`, `unlike(echoId, userId)`.
- `comments.service.ts`: `list(echoId)`, `add(echoId, userId, content)`.

Mantén **cada servicio** pequeño y testeable. Sin lógica de UI.

---

7) Guards funcionales (protección de rutas)

7.1) auth.guard.ts

- Permite pasar si hay sesión; si no, redirige a `/login`.

```
// fragmento
export const authGuard: CanActivateFn = () => {
  const auth = inject(AuthService);
  const router = inject(Router);
  return auth.isAuthenticated() ? true : router.createUrlTree(['/login']);
};
```

7.2) guest.guard.ts

- Si hay sesión, redirige a `/` (bloquea `/login` y `/signup`).

---

8) Páginas (features) con control flow moderno

8.1) FeedPage

Responsabilidades:

- Cargar página de echos desde `EchosService`.
- Estado local con **signals**: `echos`, `loading`, `page`.
- Renderizar con `@if`, `@for`, `@empty`.
- Pasar callbacks a `ui-echo-card` para like/unlike (la página ejecuta el servicio y refresca).

Template — fragmento clave:

```
<section class="container">
  <h1>Feed</h1>

  @if (loading()) {
```

```
    <p>Cargando…</p>
  } @else {
    @for (e of echos(); track e.id) {
      <ui-echo-card [echo]="e"
                    [onLike]="() => like(e.id)"
                    [onUnlike]="() => unlike(e.id)"></ui-echo-card>
    } @empty {
      <p>Sin publicaciones aún.</p>
    }
  }

  <div class="pager">
    <ui-button (click)="prev()" [disabled]="page()===0">Anterior</ui-
button>
    <ui-button (click)="next()">Siguiente</ui-button>
  </div>
</section>
```

8.2) EchoDetailPage

- Cargar eco + conteos.
- Listar comentarios con `@for`.
- Form para agregar comentario (opcional: optimismo de UI).

8.3) ProfilePage

- Cargar por `:username`.
- Si es tu perfil, mostrar inputs de edición condicional con `@if` (p.ej., botón “Guardar”).

8.4) LikedPage

- Listar echos marcados con like del usuario actual.
- Reutilizar `ui-echo-card`.

8.5) LoginPage / SignupPage

- Formularios mínimamente validados.
- Llamadas a `AuthService` y redirección posterior.

---

9) Enrutamiento (standalone + guards)

Archivo: `src/app/app.routes.ts`

Definir todas las rutas al inicio:

```
// fragmento
export const routes: Routes = [
  { path: 'login',  loadComponent: () => import('./features/auth/
login.page').then(m => m.LoginPage),  canActivate: [guestGuard] },
```

```
  { path: 'signup', loadComponent: () => import('./features/auth/
signup.page').then(m => m.SignupPage), canActivate: [guestGuard] },

  { path: '', loadComponent: () => import('./features/feed/
feed.page').then(m => m.FeedPage), canActivate: [authGuard] },
  { path: 'echo/:id', loadComponent: () => import('./features/echo-detail/
echo-detail.page').then(m => m.EchoDetailPage), canActivate: [authGuard] },
  { path: 'profile/:username', loadComponent: () => import('./features/
profile/profile.page').then(m => m.ProfilePage), canActivate: [authGuard] },
  { path: 'liked', loadComponent: () => import('./features/liked/
liked.page').then(m => m.LikedPage), canActivate: [authGuard] },

  { path: '**', redirectTo: '' },
];
```

Archivo: `src/app/app.config.ts`

```
// fragmento
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes)],
};
```

Prueba rápida:

- Sin sesión → `/` debe redirigir a `/login` (auth.guard).
- Con sesión → `/login` / `/signup` deben redirigir a `/` (guest.guard).

---

10) Conectar todo (orden sugerido)

1. **Inicializa sesión**

2. Llama `auth.initSession()` una vez al arrancar (por ejemplo en la primera página protegida o en un pequeño `AppComponent`).

3. **Supabase ping**

4. Desde `EchosService.getPage(0)`, haz una llamada y registra en consola para verificar conexión.

5. **Guards**

6. Verifica redirecciones según estado de sesión.

7. **Feed**

8. Carga echos con `@if` (loading/error) y `@for` (lista).

9. Implementa paginación mínima (`page` con signals + `getPage(page)`).
10. **Detalle & comentarios**
11. Carga eco por `:id` .
12. Lista comentarios con `@for` y agrega comentario con un form.
13. **Perfil**
14. Carga por `:username` ; si es tuyo, habilita edición condicional.
15. **Likes**
16. Acciones en la **página** (no en la card). Tras `like/unlike` refresca el item o la lista.
17. **Realtime (opcional)**
18. Si `environment.enableRealtime` es `true` , suscríbete a cambios en `likes/comments` y refresca contadores/listas.

---

11) Estilos mínimos y accesibilidad

- `styles.scss` : define utilidades `.container` , `.row` , `.muted` , `.card` , `.avatar` , `.btn` , `.pager` .
- Usa `:focus-visible` , `:disabled` y tamaño táctil ≥ 44×44 px.
- Evita *CSS leak*: estilos globales solo para utilidades; el resto por componente.

---

12) Checkpoints de aprendizaje (para evaluar avance)

- **C1 — Rutas:** todas las páginas vacías compilan y navegan.
- **C2 — Auth:** iniciar/cerrar sesión; guards responden.
- **C3 — Feed:** `@for` + `@if` , paginación mínima.
- **C4 — Crear Echo:** (composer básico en feed o modal sencillo).
- **C5 — Detalle:** conteos y comentarios; agregar comentario.
- **C6 — Perfil:** lectura y edición (si es propio).
- **C7 — Liked:** listado y “quitar like”.
- **C8 (opcional) — Realtime:** contadores/comentarios en vivo.

Apéndice A — Tips prácticos

- **Presentación vs Datos:** `shared` no conoce servicios; `features` orquesta; `core/services` accede a Supabase.
- **Signals > BehaviorSubject** para estado local de página.

- **@if/@for/@empty**: reemplaza `*ngIf/*ngFor` donde tenga sentido y aprovecha `track` en `@for`.
- **Errores/UX:** muestra estados `loading`, `error` y botones `disabled`.
- **Testing incremental:** stubs de servicios para probar plantillas y flujo.

Apéndice B — Pequeñas tareas (para los/las estudiantes)

1. Completar métodos de `AuthService` y probar login/signup/signout.
2. Implementar `EchosService.getById`, `createEcho` y uso en Feed/Detalle.
3. Implementar `CommentsService.add` con optimismo de UI.
4. Implementar `LikesService.like/unlike` y refresco del item.
5. Añadir edición de perfil si coincide con usuario actual.
6. Opcional: canal realtime para `likes/comments`.

¡Listo! Con esta guía lineal, el alumnado puede ir de 0 → app funcional usando Angular 20 moderno, sin depender de módulos, comprendiendo guards funcionales, `@if/@for` y servicios bien separados.