6 Patrones de escalado

Reparto de responsabilidades y reutilización de código.

6.1 Patrón Container/Presenter

6.1.1 Extraer presentación a un componente simple

`routes/home/activity.component`

```
# Create presentational home componente
ng g c routes/home/activity
```

- Move presentational logic

```
<div>
  <span>
    <a [routerLink]="['/','bookings', activity().slug]">{{ activity().name }}
</a>
  </span>
  <span>{{ activity().location }}</span>
  <span>{{ activity().price | currency }}</span>
  <span>{{ activity().date | date: 'dd-MMM-yyyy' }}</span>
</div>
```

- Move imports and declare input

```
{
 imports: [CurrencyPipe, DatePipe, RouterLink],
}
activity = input.required<Activity>();
```

6.1.2 Refactorizar componente contenedor inteligente

`routes/home/home.page`

- cambiar imports

```
imports: [ActivityComponent],
```

- declarar uso de componente

```
<article>
  <header>
    <h2>Activities</h2>
  </header>
  <main>
    @for (activity of activities(); track activity.id) {
    <lab-activity [activity]="activity" />
    }
  </main>
</article>
```

6.1.3 Comunicación entre contenedor y presentador

routes/home/home.page

```
<article>
  <header>
    <h2>Activities</h2>
  </header>
  <main>
    @for (activity of activities(); track activity.id) {
    <lab-activity [activity]="activity" [(favorites)]="favorites"
(favoritesChange)="onFavoritesChange($event)" />
    }
  </main>
  <footer>
    <small>
      Showing
      <mark>{{ activities().length }}</mark>
      activities, you have selected
      <mark>{{ favorites.length }}</mark>
      favorites.
    </small>
  </footer>
</article>
```

```
export class HomePage {
  // * Injected services division

  #service = inject(HomeService);

  // * Signals division

  /** The list of activities to be presented */
  activities: Signal<Activity[]> = toSignal(this.#service.getActivities$(), {
initialValue: [] });
```

```
  // * Properties division

  /** The list of favorites */
  favorites: string[] = [];

  // * Methods division

  /** Handles the change of the favorites list */
  onFavoritesChange(favorites: string[]): void {
    console.log("Favorites changed", favorites);
  }
}
```

y routes/home/activity.component

```
<div>
  <span>
    <input type="checkbox" name="" class="secondary outline"
(click)="toggleFavorite(activity().slug)" />
  </span>
  <span>
    <a [routerLink]="['/','bookings', activity().slug]">{{ activity().name }}
</a>
  </span>
  <span>{{ activity().location }}</span>
  <span>{{ activity().price | currency }}</span>
  <span>{{ activity().date | date: 'dd-MMM-yyyy' }}</span>
  <lab-activity-status [status]="activity().status" />
</div>
```

```
export class ActivityComponent {
  // * Input signals division

  /** The current Activity to be presented*/
  activity: InputSignal<Activity> = input.required<Activity>();

  // * Model signals division

  /** The list of favorites */
  favorites: ModelSignal<string[]> = model<string[]>([]);

  // * Methods division

  /** Toggles the favorite status of the given activity */
  toggleFavorite(slug: string): void {
    this.favorites.update((favorites) => {
```

```
        if (favorites.includes(slug)) {
          return favorites.filter((favorite) => favorite !== slug);
        }
        return favorites.concat(slug);
      });
    }
  }
```

6.2 Servicios e inyección de dependencias

6.2.1 Extraer lógica y datos a un servicio fachada

routes/home/home.service

```
# Create home service
ng g s routes/home/home
```

```
@Injectable({
  providedIn: "root",
})
export class HomeService {
  #http = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";

  getActivities() {
    return this.#http.get<Activity[]>(this.#apiUrl);
  }
}
```

6.2.2 Inyectar dependencias en el componente contenedor

routes/home/home.page

```
export default class HomePage {
  #service = inject(HomeService);
  activities: Signal<Activity[]> = toSignal(this.#service.getActivities(), {
initialValue: [] });
}
```

6.3 Principio DRY con código compartido

6.3.1 Servicios y utilidades de datos comunes

```
# generate activities service
ng g s shared/api/activities
# go to shared/api folder
cd shared/api
# create file api.functions.ts
touch api/signal.functions.ts
```

tsconfig.json

```
{
  "compilerOptions": {
    "baseUrl": "./",
    "paths": {
      "@api/*": ["src/app/shared/api/*"]
    }
  }
}
```

shared/api/activities.service

```
@Injectable
export class ActivitiesService {
  #http = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";

  getActivities() {
    return this.#http.get<Activity[]>(this.#apiUrl);
  }

  getActivityBySlug(slug: string | undefined) {
    if (!slug) return of(NULL_ACTIVITY);
    const url = `${this.#apiUrl}?slug=${slug}`;
    return this.#http.get<Activity[]>(url).pipe(
      map((activities) => activities[0] || NULL_ACTIVITY),
      catchError(() => of(NULL_ACTIVITY))
    );
  }

  putActivity(activity: Activity) {
    const url = `${this.#apiUrl}/${activity.id}`;
    return this.#http.put<Activity>(url, activity).pipe(
      catchError((error) => {
```

```
          console.error("Error updating activity", error);
          return throwError(() => new Error(error));
        })
      );
    }
  }
```

usarlo en `home.service` y en `bookings.page`

```
@Injectable({
  providedIn: "root",
})
export class HomeService {
  #activities = inject(ActivitiesService);

  getActivities() {
    return this.#activities.getActivities();
  }
}
```

```
export default class BookingsPage {
  #activitiesService = inject(ActivitiesService);

  activity: Signal<Activity> = toSignal(
    toObservable(this.slug).pipe(switchMap((slug) =>
this.#activitiesService.getActivityBySlug(slug))),
    { initialValue: NULL_ACTIVITY }
  );

  #updateActivityOnBookings() {
    if (!this.booked()) return;
    this.#activitiesService.putActivity(this.activity()).subscribe(() =>
console.log("Activity status updated"));
  }
}
```

`shared/api/signal.functions`

```
export type ApiTarget$<T, K> = (sourceValue: T) => Observable<K>;

export function toSignalMap<T, K>(source: Signal<T>, apiTarget$: ApiTarget$<T,
K>, initialValue: K): Signal<K> {
  const source$ = toObservable(source);
  const apiResult$ = source$.pipe(switchMap(apiTarget$));
  return toSignal(apiResult$, { initialValue });
}
```

usarlo en `bookings.page`

```
export default class BookingsPage {
  #service = inject(ActivitiesService);
  activity: Signal<Activity> = toSignalMap(
    this.slug,
    (slug) => this.#activitiesService.getActivityBySlug(slug),
    NULL_ACTIVITY
  );
}
```

6.3.2 Lógica y tipos de dominio

```
# go to shared folder
cd shared
# move domain into shared
mv domain shared
touch domain/activity.functions.ts
```

```
{
  "compilerOptions": {
    "baseUrl": "./",
    "paths": {
      "@domain/*": ["src/app/shared/domain/*"]
    }
  }
}
```

`shared/domain/activity.type`

`shared/domain/booking.type`

`shared/domain/activity.functions`

```
export function changeActivityStatus(activity: Activity, totalParticipants:
number) {
  if (["draft", "done", "cancelled"].includes(activity.status)) return;
  if (totalParticipants >= activity.maxParticipants) {
    activity.status = "sold-out";
  } else if (totalParticipants >= activity.minParticipants) {
    activity.status = "confirmed";
  }
}
```

usarlo en `bookings.page`

```
  #changeStatusOnTotalParticipants() {
    const totalParticipants = this.totalParticipants();
    changeActivityStatus(this.activity(), totalParticipants);
    this.participants.update((participants) => {
      participants.splice(0, participants.length);
      for (let i = 0; i < totalParticipants; i++) {
        participants.push({ id: participants.length + 1 });
      }
      return participants;
    });
  }
```

6.3.3 Componentes reutilizables

```
# generate activity-status component
ng g c shared/ui/activity-status
```

```
{
  "compilerOptions": {
    "baseUrl": "./",
    "paths": {
      "@ui/*": ["src/app/shared/ui/*"]
    }
  }
}
```

shared/ui/activity-state.component

```
.draft {
  color: aqua;
  font-style: italic;
}
.published {
  color: navy;
}
.confirmed {
  color: green;
}
.sold-out {
  color: teal;
  font-style: italic;
}
.done {
  color: olive;
  font-style: italic;
}
```

```
.cancelled {
  color: maroon;
  font-style: italic;
}
```

```
<span [class]="activity().status">{{ activity().status }}</span>
```

```
export class ActivityStatusComponent {
  status = input.required<ActivityStatus>();
}
```

usarlo en `bookings.page` y en `activity.component`

```
import { ActivityStatusComponent } from "@ui/activity-status";
{
  imports: [ActivityStatusComponent],
}
```

```
<lab-activity-status [status]="activity.status" />
```