7 Programación reactiva

7.1 Un almacén global basado en Signals

7.1.1 Crear un store basado en signals

```
# Generate as a normal service
ng g s shared/state/favorites-store
# Rename to FavoritesStore class in a favorites.store.ts file
# Add  "@state/*": ["src/app/shared/state/*"] to tsconfig.json
```

```
export class FavoritesStore {
  #state: WritableSignal<string[]> = signal<string[]>([]);

  count: Signal<number> = computed(() => this.#state().length);
  state: Signal<string[]> = this.#state.asReadonly();

  setState(favorites: string[]): void {
    this.#state.set(favorites);
  }
}
```

7.1.2 Uso desde páginas o componentes inteligentes

Desde la `home.page.ts`

```
export default class HomePage {
  #service = inject(HomeService);

  #favorites = inject(FavoritesStore);

  activities: Signal<Activity[]> = toSignal(this.#service.getActivities$(), {
initialValue: [] });

  favorites: string[] = this.#favorites.state();

  onFavoritesChange(favorites: string[]): void {
    console.log("Favorites changed", favorites);
    this.#favorites.setState(favorites);
  }
}
```

Desde el `header.component.ts`

```
export class HeaderComponent {
  #favorites = inject(FavoritesStore);

  title = "Activity Bookings";

  favCount = this.#favorites.count;
}
```

Renombrarlo como `header.widget` para remarcar su comportamiento inteligente

7.1.3 Persistencia y reutilización

`ng g s shared/services/platform`

```
@Injectable({
  providedIn: "root",
})
export class PlatformService {
  #platformId = inject(PLATFORM_ID);
  get isServer() {
    return isPlatformServer(this.#platformId);
  }
  get isBrowser() {
    return !this.isServer;
  }
}
```

`ng g s shared/services/local-repository`

```
export class LocalRepository {
  #platformService = inject(PlatformService);

  save(key: string, value: any): void {
    if (this.#platformService.isServer) return;
    const serialized = JSON.stringify(value);
    localStorage.setItem(key, serialized);
  }

  load<T>(key: string, defaultValue: T): T {
    if (this.#platformService.isServer) return defaultValue;
    const found = localStorage.getItem(key);
    if (found) {
      return JSON.parse(found);
    }
    this.save(key, defaultValue);
    return defaultValue;
  }

  remove(key: string): void {
    if (this.#platformService.isServer) return;
    localStorage.removeItem(key);
  }
}
```

```
ng g c routes/favorites --type=page --flat=false
```

```
@Component({
  selector: "lab-favorites",
  standalone: true,
  imports: [],
  template: `
    @for (favorite of favorites(); track favorite) {
      <div>{{ favorite }}</div>
      <hr />
    } @empty {
      <div>No favorites yet</div>
    }
  `,
  styles: ``,
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export default class FavoritesPage {
  #favorites: FavoritesStore = inject(FavoritesStore);
  favorites: Signal<string[]> = this.#favorites.state;
}
```

en el `activity.component`

```
<input
  type="checkbox"
  name=""
  class="secondary outline"
  [checked]="favorites().includes(activity().slug)"
  (click)="toggleFavorite(activity().slug)" />
```

7.2 Usando el router como almacén

7.2.1 Un widget con señales

```
ng g c shared/ui/filter --type=widget
```

```
export type SortOrders = "asc" | "desc";

export type Filter = {
  search: string;
  orderBy: string;
  sort: SortOrders;
};

export const DEFAULT_FILTER: Filter = {
  search: "",
  orderBy: "id",
  sort: "asc",
};
```

```
<form>
  <input type="search" name="search" [(ngModel)]="search"
placeholder="Search..." />
  <fieldset class="grid">
    <select name="orderBy" [(ngModel)]="orderBy" aria-label="Choose field to
sort by...">
      <option value="id">Sort by ID</option>
      <option value="name">Sort by Name</option>
      <option value="date">Sort by Date</option>
      <option value="price">Sort by Price</option>
    </select>
    <fieldset>
      <legend>Sort order:</legend>
      <input type="radio" name="sort" id="asc" value="asc" [(ngModel)]="sort" />
      <label for="asc">Ascending</label>
      <input type="radio" name="sort" id="desc" value="desc" [(ngModel)]="sort"
/>
      <label for="desc">Descending</label>
    </fieldset>
  </fieldset>
</form>
```

```
export class FilterWidget {
  search: WritableSignal<string> = signal<string>(DEFAULT_FILTER.search);
  orderBy: WritableSignal<string> = signal<string>(DEFAULT_FILTER.orderBy);
  sort: WritableSignal<SortOrders> = signal<SortOrders>(DEFAULT_FILTER.sort);

  #filter = computed(() => ({ search: this.search(), orderBy: this.orderBy(),
sort: this.sort() }));

  constructor() {
    effect(() => console.log("Current filter", this.filter()));
  }
}
```

7.2.2 Señales desde Query Params

- Escribir en los parámetros en `filter.widget`

```
export class FilterWidget {
  search: WritableSignal<string> = signal<string>(DEFAULT_FILTER.search);
  orderBy: WritableSignal<string> = signal<string>(DEFAULT_FILTER.orderBy);
  sort: WritableSignal<SortOrders> = signal<SortOrders>(DEFAULT_FILTER.sort);

  #filter = computed(() => ({ search: this.search(), orderBy: this.orderBy(),
sort: this.sort() }));

  constructor() {
    const router = inject(Router);
    effect(() => router.navigate([], { queryParams: this.#filter() }));
  }
}
```

- Recoger los valores en `home.page`

```
export default class HomePage {
  search: InputSignal<string | undefined> = input<string>();
  orderBy: InputSignal<string | undefined> = input<string>();
  sort: InputSignal<SortOrders | undefined> = input<SortOrders>();
}
```

```
<footer>
  <small>
    <span>
      Filtering by
      <mark>{{ search() }}</mark>
    </span>
    <span>
```

```
      Order by
      <mark>{{ orderBy() }} {{ sort() }}</mark>
    </span>
    <span>
      Got
      <mark>{{ activities().length }}</mark>
      activities.
    </span>
    <span>
      You have selected
      <mark>{{ favorites.length }}</mark>
      favorites.
    </span>
  </small>
</footer>
```

7.2.3 Query params observables

```
export class FilterWidget {
  #activatedRoute: ActivatedRoute = inject(ActivatedRoute);
  #filterParams$: Observable<Params> = this.#activatedRoute.queryParams;
  #defaultFilter: Signal<Params | Filter> = toSignal(this.#filterParams$, {
initialValue: DEFAULT_FILTER });

  search: WritableSignal<string> = signal<string>(this.#defaultFilter().search);
  orderBy: WritableSignal<string> = signal<string>
(this.#defaultFilter().orderBy);
  sort: WritableSignal<SortOrders> = signal<SortOrders>
(this.#defaultFilter().sort);
}
```

7_3 Operadores avanzados de RxJs

7_3_1 Observando y operando con eventos de usuario

`ng g c shared/ui/search`

```
@Component({
  selector: "lab-search",
  standalone: true,
  imports: [],
  template: `
    <input #searchInput type="search" [value]="searchTerm()"
placeholder="Search..." />
  `,
  styles: ``,
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class SearchComponent {
  // * View Signals division

  // The search input element reference signal
  #searchInputEl: Signal<ElementRef | undefined> = viewChild("searchInput", {
read: ElementRef });

  // * Model Signals division

  /** The search term model (i/o) signal */
  searchTerm: ModelSignal<string> = model<string>("");

  constructor() {
    effect(() => {
      const inputEl = this.#searchInputEl();
      if (!inputEl) return;
      // Observable from search events,
      // pipeline to clean up the input value,
      // and subscription emitting the search term signal
      fromEvent<Event>(inputEl.nativeElement, "input")
        .pipe(
          tap((event: Event) => console.log("💫 input event", event)),
          map((event: Event) => (event.target as HTMLInputElement).value),
          tap((value) => console.log("💫 input value", value)),
          filter((value) => value.length > 2),
          tap((filteredValue) => console.log("💫 input value after filter",
filteredValue)),
          debounceTime(300),
          tap((debouncedValue) => console.log("💫 input value after debounce",
debouncedValue)),
          distinctUntilChanged(),
          tap((distinctValue) => console.log("💫 input value after
distinctUntilChanged", distinctValue))
        )
        .subscribe((searchTerm) => this.searchTerm.set(searchTerm));
    });
  }
}
```

filter.widget

```
<form>
  <!-- <input type="search" name="search" [(ngModel)]="search"
placeholder="Search..." /> -->
```

```
  <lab-search [(searchTerm)]="search" />
  <fieldset class="grid">
    <select name="orderBy" [(ngModel)]="orderBy" aria-label="Choose field to
sort by...">
      <option value="id">Sort by ID</option>
      <option value="name">Sort by Name</option>
      <option value="date">Sort by Date</option>
      <option value="price">Sort by Price</option>
    </select>
    <fieldset>
      <legend>Sort order:</legend>
      <input type="radio" name="sort" id="asc" value="asc" [(ngModel)]="sort" />
      <label for="asc">Ascending</label>
      <input type="radio" name="sort" id="desc" value="desc" [(ngModel)]="sort"
/>
      <label for="desc">Descending</label>
    </fieldset>
  </fieldset>
</form>
```

7_3_2 Operadores observables de primer orden

`activities.repository.ts`

```
export class ActivitiesRepository {
  /**
   * Get all activities from the API based on a filter
   * @param filter The filter to be applied
   * @returns An observable with the activities
   */
  getActivitiesByFilter$(filter: Filter): Observable<Activity[]> {
    const url = `${this.#apiUrl}?
q=${filter.search}&_sort=${filter.orderBy}&_order=${filter.sort}`;
    return this.#http.get<Activity[]>(url);
  }
}
```

`home.service.ts`

```
export class HomeService {
  /**
   * Get all activities from the API based on a filter
   * @param partialFilter The partial filter to be applied
   * @returns An observable with the activities
   */
  getActivitiesByFilter$(partialFilter: Partial<Filter>): Observable<Activity[]>
{
    const filter: Filter = {
      search: partialFilter.search || DEFAULT_FILTER.search,
      orderBy: partialFilter.orderBy || DEFAULT_FILTER.orderBy,
      sort: partialFilter.sort || DEFAULT_FILTER.sort,
    };
    return this.activitiesRepository.getActivitiesByFilter$(filter);
  }
}
```

home.page.ts

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

  /** Computed filter from the search, orderBy and sort signals */
  #filter: Signal<Filter> = computed(() => ({ search: this.search(), orderBy:
this.orderBy(), sort: this.sort() }));
  /** The filter signal as an observable */
  #filter$: Observable<Filter> = toObservable(this.#filter);
  /** A function that returns the observable of activities based on the filter
*/
  #getActivitiesByFilter$ = (filter: Filter) =>
this.#service.getActivitiesByFilter$(filter);
  /** Pipeline to get the activities observable based on the filter observable
*/
  #filter$SwitchMapApi$: Observable<Activity[]> =
this.#filter$.pipe(switchMap(this.#getActivitiesByFilter$));
  /** The activities signal based on the filter observable */
  activities: Signal<Activity[]> = toSignal(this.#filter$SwitchMapApi$, {
initialValue: [] });
```

7_3_3 Peticiones paralelas

```
export default class FavoritesPage {
  #favoritesStore: FavoritesStore = inject(FavoritesStore);

  #activitiesRepository: ActivitiesRepository = inject(ActivitiesRepository);
  // activities: Signal<string[]> = this.#favorites.state;

  #favoriteSlugs: string[] = this.#favoritesStore.state();

  #getActivityBySlug$ = (favoriteSlug: string) =>
this.#activitiesRepository.getActivityBySlug$(favoriteSlug);

  #mapActivitiesFromSlugs$: Observable<Activity>[] =
this.#favoriteSlugs.map(this.#getActivityBySlug$);
```

```

  #activities$: Observable<Activity[]> =
forkJoin(this.#mapActivitiesFromSlugs$);

  activities: Signal<Activity[]> = toSignal(this.#activities$, { initialValue:
[] });
}
```

```
@for (activity of activities(); track activity) {
<div>
  <span><a [routerLink]="['/bookings', activity.slug]">{{ activity.name }}</a>
  <span>at {{ activity.location }} on {{ activity.date }}</span>
</div>
<hr />
} @empty {
<div>No activities yet</div>
}
```