5 Comunicaciones HTTP

Envío, recepción y manejo de datos asíncronos

5.1 Consumo de un API.

5.1.1 Lectura asíncrona de datos

```
npm i -D json-server@0.17.4 json-server-auth
npm i -D copyfiles
# package.json scripts
"api": "json-server-auth ../db/prod/d.json -r ../db/r.json",
"api:reset": "copyfiles -f ../db/reset/d.json ../db/prod && npm run api",
"api:seed": "copyfiles -f ../db/seed/d.json ../db/prod && npm run api",
npm run api
```

```
//config provider
export const appConfig: ApplicationConfig = {
  providers: [provideClientHydration(), provideHttpClient(),
provideRouter(routes, withComponentInputBinding())],
};
```

```
// Home Page
export default class HomePage {
  #http$ = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";
  activities = [];

  constructor() {
    this.#http$.get<Activity[]>(this.#apiUrl).subscribe((activities) =>
(this.activities = activities));
  }
}
```

5.1.2 Envío asíncrono de cambios

```
// Bookings Page
export default class BookingsPage {
  #http$ = inject(HttpClient);
  #activitiesUrl = "http://localhost:3000/activities";
  #bookingsUrl = "http://localhost:3000/bookings";
  onBookParticipantsClick() {
    this.booked.set(true);
    const newBooking: Booking = {
      id: 0,
      userId: 0,
      activityId: this.activity().id,
      date: new Date(),
      participants: this.newParticipants(),
      payment: {
        method: "creditCard",
        amount: this.bookingAmount(),
        status: "pending",
      },
    };
    this.#http$.post<Booking>(this.#bookingsUrl, newBooking).subscribe({
      next: () => this.#updateActivityStatus(),
      error: (error) => console.error("Error creating booking", error),
    });
  }

  #updateActivityStatus() {
    const activityUrl = `${this.#activitiesUrl}/${this.activity().id}`;
    this.#http$.put<Activity>(activityUrl, this.activity()).subscribe({
      next: () => console.log("Activity status updated"),
      error: (error) => console.error("Error updating activity", error),
    });
  }
}
```

5.2 Asincronismo y señales

5.2.1 Señales con los datos recibidos

```
// Home Page
export default class HomePage {
  #http$ = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";
  activities = signal<Activity[]>([]);

  constructor() {
    this.#http$.get<Activity[]>(this.#apiUrl).subscribe((activities) =>
this.activities.set(activities));
  }
}
```

5.2.2 Señales para enviar cambios

Usando efectos para obtener datos asíncronos

```
// Bookings Page
export default class BookingsPage {
  #http$ = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";
  slug = input<string>();

  activity = signal<Activity>(NULL_ACTIVITY);

  constructor() {
    effect(() => this.#getActivityOnSlug(), { allowSignalWrites: true });
  }

  #getActivityOnSlug() {
    const activityUrl = `${this.#activitiesUrl}?slug=${this.slug()}`;
    this.#http$.get<Activity[]>(activityUrl).subscribe((activities) => {
      this.activity.set(activities[0] || NULL_ACTIVITY);
    });
  }
}
```

5.3 Operadores RxJS.

5.3.1 Tuberías funcionales

```
export default class BookingsPage {
  #getActivityOnSlug() {
    const activityUrl = `${this.#activitiesUrl}?slug=${this.slug()}`;
    this.#http$
      .get<Activity[]>(activityUrl)
      .pipe(
        map((activities: Activity[]) => activities[0] || NULL_ACTIVITY),
        catchError((error) => {
          console.error("Error getting activity", error);
          return of(NULL_ACTIVITY);
        })
      )
      .subscribe((activity: Activity) => {
        this.activity.set(activity);
      });
  }
}
```

5.3.2 Interoperabilidad básica de señales y observables

```
export default class HomePage {
  #httpClient$: HttpClient = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";
  activities: WritableSignal<Activity[]> = signal<Activity[]>([]);

  constructor() {
    this.#httpClient$.get<Activity[]>(this.#apiUrl).subscribe((result) =>
this.activities.set(result));
  }
}
```

```
import { toSignal } from "@angular/core/rxjs-interop";
export default class HomePage {
  #httpClient$: HttpClient = inject(HttpClient);
  #apiUrl = "http://localhost:3000/activities";
  // What toSignal() do for us?
  // 1 - subscribe
  // 2 - signal.set(result)
  // 3 - unsubscribe from observable
  // 4 - returned signal is read-only, no mutable elsewhere
  activities: Signal<Activity[]> = toSignal<Activity[]>(
    this.#httpClient$.get<Activity[]>(this.#apiUrl).pipe(catchError(() =>
of([]))),
    { initialValue: [] }
  );
}
```

5.3.3 Usos avanzados de rxjs-interop

```
import { toObservable, toSignal } from "@angular/core/rxjs-interop";
export default class BookingPage {
  #apiUrl = "http://localhost:3000/activities";

  /** The slug of the activity that comes from the router */
  slug: InputSignal<string> = input.required<string>();

  // 1 -> Convert source signal to an observable
  slug$: Observable<string> = toObservable(this.slug);
  // 2 -> RxJs operators do the heavy work with other async calls and
transformations
  activity$: Observable<Activity> = this.slug$.pipe(
    switchMap((slug: string) => {
      const url = `${this.#apiUrl}?slug=${slug}`;
      return this.#http.get<Activity[]>(url);
    }),
    map((activities: Activity[]) => activities[0])
  );
  // 3 - > Convert back the observable into a signal usable from the template
  activity: Signal<Activity> = toSignal(this.activity$, { initialValue:
NULL_ACTIVITY });

  // 4 - > Remove effects!
}
```