# 5 Comunicaciones HTTP

Envío, recepción y manejo de datos asínconos

## 5.1 Consumo de un API.

### 5.1.1 Lectura asíncrona de datos

```
npm i -D json-server json-server-auth
npm i -D copyfiles
#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

```
// 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 de señales y observables

```
import { toObservable, toSignal } from "@angular/core/rxjs-interop";

simpleSignal: Signal<string> = toSignal(of("Angular"), { initialValue: "" });
simpleObs: Observable<string> = toObservable(this.simpleSignal);
complexSignal: Signal<string> = toSignal(of(this.slug), { initialValue: "" });

// Original implementation
// activity = signal<Activity>(NULL_ACTIVITY);
// With effect
// constructor() {
//     const ALLOW_WRITE = { allowSignalWrites: true };
//     effect(() => this.#getActivityOnSlug(), ALLOW_WRITE);
// }

// Alternative implementation using toSignal and toObservable and switchMap

activity: Signal<Activity> = toSignal(
  toObservable(this.slug).pipe(
    switchMap((slug) =>
      this.#http$.get<Activity[]>(`${this.#activitiesUrl}?slug=${slug}`).pipe(
        map((activities) => activities[0] || NULL_ACTIVITY),
        catchError(() => of(NULL_ACTIVITY))
      )
    )
  ),
  { initialValue: NULL_ACTIVITY }
);
```