



Clase 7

Comunicación con APIs REST
mediante HttpClient

¿Qué es HttpClient?

HttpClient es el servicio oficial de Angular para comunicarse con APIs REST

```
fetch('/api/team-members')  
  .then((response) => response.json())  
  .then((data) => console.log(data))  
  
// Angular provee HttpClient con superpoderes 🚀  
this.http  
  .get<TeamMember[]>('/api/team-members')  
  .subscribe((members) => console.log(members))
```

Beneficios: Tipado fuerte, Observables, Interceptores, Testing fácil

JSON Server (Backend Local)

Servidor REST automático con un archivo JSON

```
# Instalar  
npm install -D json-server  
  
# Crear db.json  
{  
  "team-members": [  
    { "id": "a1b2", "name": "Ana" }  
  ]  
}  
  
# Ejecutar  
npx json-server db.json --port 3000
```

Rutas Automáticas de JSON Server

Con `db.json`:

```
{  
  "team-members": [...]  
}
```

JSON Server crea automáticamente:

GET	/team-members	→ Lista todos
GET	/team-members/1	→ Obtiene el id 1
POST	/team-members	→ Crea uno nuevo
PUT	/team-members/1	→ Actualiza el id 1
DELETE	/team-members/1	→ Elimina el id 1

Buzón Mágico (Observables)

Imagina que hay un buzón mágico frente a tu casa. ¡En este buzón pueden aparecer diferentes mensajes!

 Buzón Mágico (Observable)

↓

Esperando...

↓

Mensaje...

Observable

El Observable es como ese buzón mágico. Es un lugar donde los mensajes pueden llegar con el paso del tiempo.

Subscribirse = Abrir la puerta del buzón

Cuando haces `.subscribe()`, es como si abrieras la puerta del buzón y dijeras:

```
// Abres la puerta del buzón  
mailbox.subscribe((message) => {  
  console.log('Recibí un mensaje:', message)  
})
```

Los mensajes llegan uno por uno

- 🚗 Un carro rojo llega a las 9am
- ⚽ Una pelota llega a las 11am
- ✉ Una carta llega a las 2pm



Operators

tap - Mirar sin modificar:

"Cada mensaje que llegue, míralo y registra qué llegó"

 → Log: "¡Llegó un carro!" →  (sigue igual)

 → Log: "¡Llegó una pelota!" →  (sigue igual)

takeUntil - Cierra el buzón cuando cierta señal llega:

"Recibe mensajes hasta que suene la alarma 

  Mensaje recibido

  Mensaje recibido

 ¡Alarma! → Buzón cerrado, no más mensajes

Más Operators

startWith - Empieza con un mensaje inicial:

"Antes de esperar mensajes nuevos, aquí tienes uno para empezar"
✉ (mensaje inicial) → Luego esperamos más...

catchError - Maneja problemas:

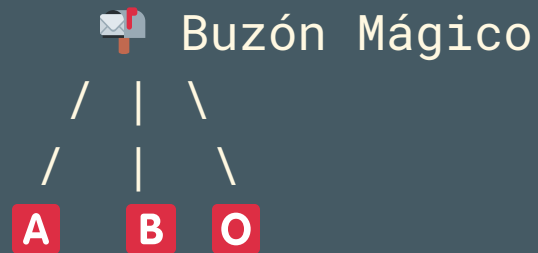
"Si hay un problema, haz algo especial"
✗ Error al traer mensaje → 📺 Envía un mensaje de repuesto

El Flujo Completo

```
📧 Buzón Mágico (Observable)
↓
✉ Empieza con una carta ← startWith
↓
🚗 Llega un carro nuevo
↓
📝 Registramos que llegó ← tap (logging)
↓
🔔 Alarma suena ← takeUntil (cleanup)
↓
🚫 Buzón cerrado
↓
"Terminamos de recibir mensajes" ← complete
```

Múltiples Observadores

¡Puedes tener muchos subcriptores viendo el mismo buzón!



Cuando llega un mensaje 🚗, TODOS lo ven al mismo tiempo.

Observables vs Promesas


```
// Promesa - Solo 1 valor  
const promesa = fetch('/api/data').then((data) => console.log(data)) // 1 vez  
  
// Observable - Puede emitir múltiples valores  
const observable = this.searchControl.valueChanges.subscribe((valor) =>  
  console.log(valor)  
) // Cada vez que escribas
```

Observable = Stream de datos que puede emitir 0, 1, o N valores

HttpClient en Angular

Paso 1: Configurar en app.config.ts

```
import { provideHttpClient } from '@angular/common/http'

export const appConfig: ApplicationConfig = {
  providers: [
    provideHttpClient(), //  Habilita HttpClient
  ],
}
```

Paso 2: Inyectar en el servicio

```
constructor(private http: HttpClient) {}
```

GET / POST / PUT / DELETE

```
// Get - Obtener todos
this.http.get<TeamMember[]>('/api/team-members')
// Get - Obtener uno
this.http.get<TeamMember[]>(`/api/team-members/${id}`)
// POST - Crear
this.http.post<TeamMember>('/api/team-members', member);
// PUT - Actualizar
this.http.put<TeamMember>(`/api/team-members/${id}`, member);
// DELETE - Eliminar
this.http.delete<void>(`/api/team-members/${id}`);
```

Todas retornan Observables - usar `.subscribe()` para ejecutar

Interceptores - Middleware HTTP

Código que se ejecuta en TODAS las peticiones HTTP

```
export const loggingInterceptor: HttpInterceptorFn = (req, next) => {  
  console.log(`→ ${req.method} ${req.url}`)  
  
  return next(req).pipe(  
    tap((event) => {  
      if (event instanceof HttpResponse) {  
        console.log(`← ${event.status} ${req.url}`)  
      }  
    })  
  )  
}
```

Component



```
this.http.get('/api/data')
```



1. INTERCEPTOR
(Request)

→ Log: "→ GET /api/data"
→ Agrega headers



[Internet]



2. INTERCEPTOR
(Response)

→ Log: "← 200 OK (120ms)"
→ Maneja errores



```
.subscribe()
```


Registrar Interceptores

```
// app.config.ts  
import { provideHttpClient, withInterceptors } from '@angular/common/http'  
  
export const appConfig: ApplicationConfig = {  
  providers: [provideHttpClient(withInterceptors([loggingInterceptor]))],  
}
```

Orden importa: se ejecutan de arriba hacia abajo

Casos de Uso de Interceptores

```
// Logging  
console.log(`→ ${req.method} ${req.url}`)
```

```
// Autenticación  
req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })
```

```
// Manejo de Errores  
catchError((err) => {  
    /* mostrar mensaje global */  
}))
```