Promeses

Moltes de les interaccions que es fan a un front són de forma asyncrona (un event al pitjar un botó, la petició de dades a l'api...). La necessitat de l'asyncronia resideix en que no podem bloquejar la pàgina cada vegada que realitzam una acció. Si l'usuari pitja un botó per a carregar dades i aquestes tarden o no arriben mai no pot quedar la pàgina bloquejada a l'espera, l'usuari ha de poder seguir manipulant la resta d'elements.

Una de les formes de JavaScript de tractar això són les promeses. Una promesa és un "objecte" que en algun moment emetrà un valor. Hem de lligar una funció (o vàries) a aquesta promesa i s'executaran de forma asyncrona una vegada la promesa es compleixi i arribi el valor.

L'exemple més clar el podem trobar a les cridades a l'API

```
getNomina(nominaId: number): Promise<AxiosResponse> {
    return axios.get(`/nomina/${nominaId}`);
}
```

Podem observar que l'objecte que retorna és una Promesa i el valor promès serà un AxiosResponse

ES6: then/catch

Una forma de gestionar les promeses és la semblant a JQuery

```
nominaService.getNomina(1)
    .then(data) {
        // Obtenim el valor de la promesa
    }
    .catch(err) {
        // Ha ocorregut un error i no s'ha pogut satisfer la promesa
    }
    .finally() {
        // S'executarà tant si s'ha satisfet com si ha ocorregut un error
    }
```

Una problemàtica del sistema anterior és la claretat visual en l'ordre d'execució

```
// 1
nominaService.getNomina(1)
    .then(data) {
        // 2
    }
// 3
```

L'ordre d'execució probable seria: 1, 3, 2 ja que 2 s'executarà de forma asyncrona en un moment indeterminat. El codi es segueix executant ja que no queda bloquejat.

ES7: async/await

Disposam d'una forma més amigable o lineal de gestionar les promeses

```
// 1
try {
   const data = await nominaService.getNomina(1);
      // Obtenim el valor de la promesa
      // 2
}.catch(err) {
      // Ha ocorregut un error i no s'ha pogut satisfer la promesa
}.finally() {
      // S'executarà tant si s'ha satisfet com si ha ocorregut un error
}
// 3
```

En aquest cas l'ordre d'execució seria 1, 2, 3. El codi quedaria bloquejat a l'espera del resultat.

NOTA: indicar que per poder usar await a una promesa el mètode pare ha de ser async:

```
async function carregaNomina() {
   try {
      const data = await nominaService.getNomina(1);
   }
}
```

No hi ha una millor forma de gestionar les promeses, en funció del nostre codi haurem de decidir quina és la millor opció.

La primera opció (then) és menys clara però no ens bloqueja el codi, la segona és més clara però bloqueja l'execució.