Appunti – 18 Febbraio 2026

Destrutturazione in JavaScript

La destrutturazione è un meccanismo di JavaScript che permette di estrarre più proprietà da un oggetto in una singola istruzione, assegnandole direttamente a variabili locali i cui nomi corrispondono a quelli delle proprietà stesse. Anziché scrivere tre assegnazioni separate del tipo `title = req.body.title`, `description = req.body.description` e `date = req.body.date`, è possibile condensare tutto in una riga: `const { title, description, date } = req.body`. Il risultato è identico — si ottengono tre variabili pronte all'uso — ma il codice risulta più compatto e leggibile.

La sintassi si interpreta nel seguente modo: le parentesi graffe a sinistra dell'operatore di assegnazione indicano che si sta "aprendo" l'oggetto per prelevarne selettivamente i campi di interesse, identificati per nome. JavaScript ricerca all'interno di `req.body` le proprietà che si chiamano esattamente `title`, `description` e `date`, e le assegna alle variabili corrispondenti. Se una proprietà non è presente nell'oggetto sorgente, la variabile risultante assumerà semplicemente il valore `undefined`.

Nel contesto dell'endpoint POST analizzato, `req.body` rappresenta il corpo della richiesta HTTP proveniente dal client e contiene i dati del nuovo show da inserire. Grazie alla destrutturazione, è possibile lavorare direttamente con `title`, `description` e `date` nel resto del metodo, evitando di ripetere ogni volta il prefisso `req.body`.

Spread Operator in JavaScript

Lo spread operator, rappresentato dai tre punti `...`, è un meccanismo di JavaScript che consente di espandere un oggetto esistente copiando tutte le sue proprietà all'interno di un nuovo oggetto letterale. Nel codice esaminato, la risposta al client viene costruita come `{ id: info.lastInsertRowid, ...req.body }`: JavaScript crea un oggetto nuovo, gli assegna la proprietà `id` con il valore dell'identificativo appena generato dal database, e vi copia tutte le proprietà già

presenti in `req.body`, ovvero `title`, `description` e `date`. Il risultato finale è un unico oggetto che contiene tutti i dati insieme.

È fondamentale comprendere che questa operazione non modifica `req.body`: si crea un oggetto completamente nuovo che ne costituisce una copia arricchita. Questo approccio è utile perché il client che ha effettuato la richiesta POST ha trasmesso i dati dello show ma non conosce ancora l'identificativo che il database gli ha assegnato. Restituendo l'oggetto completo con l'`id` incluso, il client viene messo nelle condizioni di sapere esattamente come il record è stato persistito.

Service e Dependency Injection in Angular

Un componente Angular ha il compito di gestire la parte visiva, ossia il template HTML, la logica di presentazione nel file TypeScript e lo stile CSS. Quando però un componente necessita di svolgere operazioni che vanno oltre la semplice visualizzazione — come comunicare con un backend o condividere dati con altre parti dell'applicazione — entra in gioco il concetto di Service.

Un Service in Angular è una classe ordinaria decorata con `@Injectable`, priva di qualsiasi elemento visivo: non possiede template né stili, ed esiste esclusivamente per contenere logica riutilizzabile. Le chiamate HTTP, in particolare, non devono mai essere effettuate direttamente all'interno dei componenti, ma appartengono esclusivamente al livello dei servizi.

Il parallelismo con Spring è significativo: così come in Spring un `@Service` è un bean gestito dal container e iniettabile tramite `@Autowired`, in Angular un `@Injectable` è un oggetto gestito dal framework che può essere richiesto attraverso il costruttore del componente. In entrambi i casi si applica il principio della Dependency Injection: non è lo sviluppatore a istanziare l'oggetto con `new`, ma è il framework che se ne occupa e lo consegna già pronto.

Interfaccia Show e HttpClient

Prima di effettuare chiamate HTTP è necessario definire un modello che rappresenti la struttura dei dati da trattare. In Angular questo si fa tramite un'interfaccia TypeScript, che svolge un ruolo analogo a quello di una classe in Java, con la differenza che in TypeScript un'interfaccia definisce esclusivamente

la forma del dato senza contenere logica. L'interfaccia `Show` è stata quindi definita con i campi `id` (opzionale), `title`, `description` e `date`.

Per effettuare chiamate HTTP, Angular mette a disposizione la libreria `HttpClient`, che va importata e iniettata all'interno del servizio tramite la funzione `inject`. Un servizio può a sua volta utilizzare altri servizi: in questo caso, `ShowService` incorpora `HttpClient` per delegare ad esso la comunicazione con il backend.

Componente e Template HTML

Una volta definito il servizio, si procede alla creazione del componente. Il template HTML del componente espone un form con tre campi di input — `title`, `description` e `date` — legati alle proprietà dell'oggetto `show` tramite la direttiva `[(ngModel)]`, che in Angular realizza il two-way data binding. Il componente TypeScript deve iniettare il servizio nel proprio costruttore e, al momento del salvataggio, richiamare il metodo appropriato del servizio attraverso il gestore dell'evento `(click)` associato al pulsante.