Lezione del 09 febbraio 2026

Integrazione tra API e JavaScript

A questo punto è necessario collegare le API del backend al codice JavaScript del frontend. I dati devono essere caricati dalle API attraverso richieste HTTP. Ciò che arriverà dal server saranno oggetti SanctionDTO, non entità Sanction complete. Si riceverà quindi lo stato delle sanzioni formattato secondo la struttura definita nel DTO, contenente esattamente le informazioni che si è scelto di esporre.

Caricamento dei Dati tramite Fetch

Per recuperare i dati viene utilizzata una chiamata fetch a cui viene fornito l'indirizzo completo dell'endpoint API. La richiesta carica i dati delle sanzioni dal server. Ipotizzando che la richiesta vada a buon fine, si procede con l'estrazione del JSON dalla risposta. Quando l'operazione ha successo viene dichiarata una variabile let sanctions inizializzata come array vuoto, che conterrà le sanzioni ricevute dal server.

Rendering delle Sanzioni

Da ogni sanction ricevuta viene ricavato lo stato necessario per creare il componente. Viene dichiarata una variabile let res inizializzata come stringa vuota. Attraverso un ciclo si scorrono tutte le sanzioni contenute nell'array. Per ciascuna sanzione si crea un'istanza della classe Sanction passando l'oggetto ricevuto come stato, si invoca il metodo render su tale istanza e si accoda il risultato HTML alla variabile res. Al termine del ciclo la stringa res contiene l'HTML completo di tutte le sanzioni, pronto per essere inserito nel DOM attraverso innerHTML. In questo modo i dati provenienti dal backend vengono trasformati in componenti visuali e renderizzati nella pagina.

CONCLUSIONI

Quando si lavora con Spring è possibile definire relazioni di parentela tra entità attraverso l'uso di annotazioni specifiche che descrivono la cardinalità delle

associazioni. Un esempio tipico è la relazione uno a molti, che viene dichiarata mediante annotation che istruiscono il framework su come gestire il legame tra due classi del dominio e come tradurre questa relazione in strutture persistenti sul database.

Tuttavia, la rappresentazione interna delle entità non coincide quasi mai con quella richiesta dai consumatori esterni dell'applicazione. Per questo motivo le entità non vengono mai esposte direttamente ma subiscono un processo di mappatura verso oggetti DTO. La molteplicità dei Data Transfer Object è una caratteristica fondamentale dell'architettura applicativa, poiché non esiste un'unica rappresentazione universale dei dati. Spesso emerge la necessità di fornire viste differenti delle stesse informazioni a seconda del contesto d'uso, dell'utente che effettua la richiesta o delle operazioni che devono essere eseguite.

Queste diverse proiezioni dello stesso dato vengono realizzate attraverso DTO specializzati, ciascuno contenente solo il sottoinsieme di attributi rilevante per lo scenario specifico. La responsabilità di effettuare la trasformazione da entità a DTO ricade su componenti dedicati chiamati mapper, che implementano la logica di conversione isolandola dal resto del sistema. Il mapper riceve un'entità del dominio e produce il corrispondente oggetto di trasferimento applicando le regole di mappatura definite, garantendo così che ogni interlocutore riceva esattamente la rappresentazione dei dati più appropriata alle sue esigenze.

La creazione di un endpoint per recuperare i cittadini tramite un metodo annotato con `@GetMapping` che restituisce direttamente il risultato di `repository.findAll()` genera un problema critico nell'applicazione. Il sistema entra in un ciclo infinito durante la serializzazione JSON delle entità. Questo comportamento si manifesta perché le relazioni bidirezionali tra `Citizen` e `Sanction` creano una catena ricorsiva: ogni cittadino contiene una lista di sanzioni, e ogni sanzione contiene un riferimento al cittadino che l'ha ricevuta, innescando un processo di serializzazione che non termina mai.

La soluzione a questo problema risiede nell'utilizzo sistematico dei Data Transfer Object anche per l'entità `Citizen`. Introducendo un `CitizenDTO` si interrompe la circolarità perché questo oggetto non replica la struttura relazionale delle entità del dominio. All'interno del `CitizenDTO` non deve essere presente una lista di

oggetti `Sanction` del dominio, ma piuttosto una collezione di `SanctionDTO`, ovvero rappresentazioni semplificate delle sanzioni che non contengono riferimenti circolari al cittadino.

Per gestire la trasformazione diventa necessario implementare anche un `CitizenDTOMapper`, un componente dedicato che si occupa di convertire un'entità `Citizen` nel corrispondente `CitizenDTO`. Questo mapper applica la logica di mappatura non solo agli attributi semplici del cittadino ma anche alla collezione di sanzioni associate, trasformando ciascuna `Sanction` nel rispettivo `SanctionDTO` attraverso la collaborazione con il mapper delle sanzioni, garantendo così una conversione completa e priva di riferimenti ciclici.