Backend - 02-09-26

Ora passiamo al backend. Il file `application.properties` viene utilizzato come file di configurazione di Spring.

Per quanto riguarda le entità, si comincia dall'entità padre. Viene creata una nuova classe `Citizen` all'interno del package `com.generation.sanctionweb.model.entities`. Questa entità contiene come attributi un identificativo intero, il nome, il cognome e il codice fiscale. Per tutti gli attributi vengono generati i relativi getter e setter.

Successivamente si crea l'entità `Sanction`, che presenta come attributi un identificativo intero, una data di emissione di tipo `LocalDate`, una data di pagamento sempre di tipo `LocalDate`, lo stato della sanzione, l'importo della multa e la motivazione. Anche in questo caso vengono generati i getter e i setter per ciascun attributo.

All'interno dell'entità `Citizen` è necessario creare una lista di sanzioni, dichiarata come `List<Sanction> sanctions`. Questa relazione viene annotata con `@OneToMany` perché il cittadino rappresenta l'oggetto padre. Tuttavia, la relazione deve essere completata anche nell'altra entità, quindi è necessario definire correttamente il collegamento corrispondente all'interno dell'entità `Sanction`.
Viene completata l'entità `Sanction`. La classe è annotata con `@Entity` per indicare che rappresenta una tabella del database. L'identificativo è definito come chiave primaria tramite `@Id` e viene generato automaticamente con strategia `GenerationType.IDENTITY`.

Viene poi definita la relazione con l'entità `Citizen`. La sanzione è collegata a un singolo cittadino tramite una relazione `@ManyToOne` con caricamento di tipo `EAGER`. L'annotazione `@JoinColumn` specifica il nome della colonna `citizen_id` nel database, che funge da chiave esterna e permette di collegare ogni sanzione al relativo cittadino.

L'entità contiene inoltre gli attributi relativi alla sanzione: la data di emissione, la data di pagamento, lo stato, la motivazione e l'importo. Tutti questi campi sono gestiti tramite i rispettivi getter e setter, consentendo l'accesso e la modifica dei dati in modo controllato.

Configurazione delle Relazioni tra Entità

Dopo aver inserito le annotation sulle entità, Spring viene informato della presenza di relazioni tra gli oggetti del modello. Le annotation comunicano al framework che determinati oggetti sono collegati tra loro attraverso relazioni di tipo padre-figlio. Quando viene caricata un'entità padre, Spring si occupa automaticamente di caricare anche le entità figlie correlate, gestendo la navigazione delle associazioni in modo trasparente. Una volta completata la definizione delle entità con le relative annotation, si procede con la creazione delle repository.

Creazione delle Repository

Le repository vengono create estendendo l'interfaccia JpaRepository, specificando tra parentesi angolari il tipo dell'entità gestita e il tipo della chiave primaria. Nel caso della sanzione si scrive extends JpaRepository<Sanction,Integer>, dove Sanction rappresenta l'entità e Integer il tipo dell'identificativo. Questa estensione è sufficiente per ottenere tutti i metodi di base necessari alle operazioni CRUD senza dover scrivere alcuna implementazione. JpaRepository fornisce automaticamente metodi come save, findById, findAll, delete e molti altri, rendendo immediatamente operativa la persistenza dei dati.

Creazione dell'API REST

A questo punto il database esiste e le repository sono pronte per interagire con esso. Il passo successivo consiste nello scrivere l'API che esporrà le funzionalità del sistema attraverso endpoint HTTP. Viene creata una nuova classe nel package com.generation.sanctionweb.api denominata SanctionAPI. Questa classe viene marcata con l'annotation @RestController, che indica a Spring che si tratta di un controller REST responsabile di gestire richieste HTTP e restituire risposte in formato JSON.

All'interno della classe SanctionAPI è necessario disporre di un'istanza della repository per poter accedere ai dati. L'istanza viene ottenuta attraverso l'annotation @Autowired applicata alla dichiarazione della repository, delegando a Spring il compito di iniettare automaticamente la dipendenza. Per definire un indirizzo di base comune a tutti i metodi della classe viene utilizzata un'annotation a livello di classe che specifica il percorso /sanction/api/sanctions. Tutti gli

endpoint definiti all'interno della classe SanctionAPI saranno quindi accessibili a partire da questo percorso base, consentendo di organizzare le API in modo coerente e strutturato.

L'analisi del JSON restituito dalla chiamata API rivela una discrepanza strutturale rispetto alle esigenze del sistema. Il campo `citizen` viene ricevuto come oggetto complesso contenente attributi multipli, mentre la struttura dati attesa prevede che questo campo sia rappresentato come una semplice stringa identificativa. Questa incongruenza richiede un processo di mappatura che trasformi la rappresentazione ricevuta in una forma compatibile con i requisiti funzionali dell'applicazione.

Il concetto fondamentale che emerge è quello della traduzione dei dati tra sistemi eterogenei. Il frontend comunica secondo un protocollo definito e si aspetta informazioni strutturate in un formato specifico. Raramente le entità persistite nel database possono essere trasmesse direttamente attraverso la rete senza subire trasformazioni. Questo principio si materializza nell'utilizzo dei Data Transfer Object, abbreviati in DTO, che rappresentano oggetti specificamente progettati per il trasporto di informazioni tra strati applicativi diversi.

La distinzione cruciale risiede nel fatto che l'entità `Sanction`, così come esiste nel dominio applicativo, non possiede la forma richiesta dal client. Diventa quindi necessario introdurre un meccanismo di conversione che generi un `SanctionDTO`, ovvero una versione modificata della sanzione originale che rispetti esattamente le aspettative del destinatario. Attraverso questa mappatura si garantisce che durante la trasmissione in rete viaggino esclusivamente le informazioni necessarie, strutturate nel formato concordato, senza esporre dettagli implementativi interni o includere dati superflui che non competono al consumatore del servizio.

Data Transfer Object e Rappresentazioni Multiple

Per ogni classe del modello possono esistere molteplici Data Transfer Object, poiché è possibile decidere di inviare rappresentazioni diverse dello stesso oggetto a seconda del contesto o del destinatario. Se si inviasse l'entità Sanction così come è strutturata nel database, verrebbero trasmessi anche dati sensibili come il codice fiscale o informazioni che non si desidera esporre pubblicamente.

Per questo motivo viene creata una nuova classe nel package com.generation.sanctionweb.api.dto.

Il termine DTO indica che si stanno creando oggetti non pensati per essere utilizzati internamente dall'applicazione ma esclusivamente per essere inviati o ricevuti attraverso le API. Un DTO rappresenta la forma che un oggetto assume quando viene trasmesso, sia in fase di invio che di ricezione.

Processo di Trasformazione dei Dati

Il processo completo prevede diverse fasi. Quando arriva una richiesta HTTP, il backend recupera i dati dal database attraverso le repository. Prima di inviare questi dati al client, vengono trasformati nella loro rappresentazione DTO. Non si inviano i dati grezzi così come sono memorizzati perché potrebbero contenere troppi dati, informazioni riservate o perché si desidera modificarne la struttura per adattarla alle esigenze del frontend. Ciò che viene trasmesso attraverso la rete è sempre una trasformazione dell'oggetto originale.

Struttura del SanctionDTO

Il SanctionDTO deve contenere le informazioni essenziali per la visualizzazione della sanzione. Include la data dell'infrazione, la data di pagamento, lo status, la ragione della multa e l'importo. A differenza dell'entità Sanction, il DTO non contiene l'intero oggetto Citizen ma solamente una stringa citizen che concatena nome e cognome del cittadino, oltre a un citizenId che rappresenta l'identificativo numerico. Questo approccio evita di trasmettere l'intera struttura dell'oggetto cittadino con tutti i suoi dettagli, riducendo la quantità di dati trasferiti e limitando l'esposizione di informazioni potenzialmente sensibili.

Implementazione del DTO

Per il SanctionDTO vengono preparati i metodi getter e setter necessari ad accedere e modificare le proprietà. Un DTO non viene creato direttamente ma esce da un componente specifico chiamato mapper. Viene quindi creata una nuova classe denominata SanctionDTOMapper, il cui unico compito è trasformare oggetti Sanction in oggetti SanctionDTO e viceversa. La seconda operazione di conversione da DTO a entità non viene implementata immediatamente.

Metodo di Mappatura

Il mapper implementa un metodo pubblico denominato toDTO che accetta come parametro un oggetto Sanction e restituisce un oggetto SanctionDTO. Il metodo riceve una sanction, che è un oggetto destinato all'uso locale nel server, e produce una sanctionDTO, che è un oggetto pronto per essere inviato al client. All'interno del metodo viene istanziato un nuovo oggetto SanctionDTO assegnato alla variabile res. Successivamente vengono impostate tutte le proprietà dell'oggetto attraverso i setter. Si esegue res.setId passando il valore di sanction.getId, si impostano data, status, ragione e importo seguendo lo stesso schema. Per popolare le proprietà relative al cittadino si utilizza res.setCitizenId passando sanction.getCitizen().getId per ottenere l'identificativo, mentre per la stringa citizen si concatenano nome e cognome recuperati attraverso sanction.getCitizen().getFirstname e gli altri metodi corrispondenti. In questo modo viene completata la trasformazione da oggetto Sanction a oggetto SanctionDTO.

Il mapper deve anche essere in grado di trasformare una lista di sanzioni in una lista di DTO, implementando un metodo analogo che opera su collezioni anziché su singoli oggetti.

Annotation e Dipendenze

Nel DTO non vengono inserite annotation JPA poiché questi oggetti non devono essere persistiti nel database. Il mapper rappresenta una dipendenza che dovrà essere iniettata nei controller che ne hanno bisogno. Per rendere il mapper disponibile nel contesto di Spring viene applicata l'annotation @Service alla classe SanctionDTOMapper. Questa annotation comunica a Spring che la classe deve essere istanziata e inserita nel context applicativo, rendendola disponibile per l'iniezione automatica attraverso @Autowired. Senza questa annotation il meccanismo di dependency injection non funzionerebbe e il mapper non sarebbe utilizzabile nei controller.