2-Appunti 10-02-26 Pomeriggio Progetto WebClinic

Inserimento Paziente con Pattern DTO

Nel contesto di un'applicazione Spring Boot, quando si parla di inserimento di un paziente non si utilizza direttamente l'entità Patient ma si ricorre al pattern DTO, dove DTO sta per Data Transfer Object.

Definizione DTO → DTO è l'acronimo di Data Transfer Object e indica un oggetto utilizzato per trasferire dati tra il client e il server attraverso il corpo della richiesta HTTP.

Il termine tecnico che descrive questo processo di conversione dei dati è *serializzazione*, e per effettuare la conversione bidirezionale tra DTO ed entità si utilizza un mapper, la mappatura è una forma di serializzazione.

Annotazioni Spring Boot Fondamentali

@PostMapping

L'annotazione `@PostMapping` indica la creazione di una nuova risorsa e corrisponde operativamente a un'istruzione INSERT nel database.

Definizione @PostMapping :: `@PostMapping` è un'annotazione Spring che mappa un metodo del controller a una richiesta HTTP di tipo POST, indicando la creazione di una nuova risorsa nel sistema.

@RequestBody

L'annotazione `@RequestBody` segnala che il corpo della richiesta HTTP, contenente un formato JSON, deve essere convertito automaticamente in un oggetto Java del tipo specificato.

Definizione @RequestBody :: `@RequestBody` è un'annotazione Spring che indica che il corpo della richiesta HTTP deve essere convertito nell'oggetto Java specificato come parametro del metodo, tipicamente attraverso un processo di deserializzazione JSON.

@ResponseEntity

L'annotazione `@ResponseEntity`, sebbene opzionale, risulta fondamentale quando si desidera avere un controllo granulare sulla risposta HTTP restituita al client.

Definizione @ResponseEntity :: `@ResponseEntity` è un'annotazione Spring che permette di controllare completamente la risposta HTTP, inclusi il codice di stato, le intestazioni e il corpo della risposta.

Se si restituisse direttamente un oggetto PatientDTO senza utilizzare ResponseEntity, la risposta sarebbe sempre un codice 200, il che non permette di comunicare situazioni di errore come richieste non valide.

Conversione tra DTO ed Entità

È importante sottolineare che il database interagisce esclusivamente con le entità JPA e non con i DTO. Di conseguenza, il metodo save del repository accetta un'entità e non un DTO.

Definizione entità JPA :: Un'entità JPA è un oggetto Java mappato su una tabella del database attraverso le annotazioni della Java Persistence API, utilizzato per rappresentare e gestire i dati persistenti dell'applicazione.

Il mapper deve quindi trasformare il DTO ricevuto in un'entità prima di procedere con il salvataggio nel database.

Definizione toEntity :: `toEntity` è un metodo del mapper che converte un DTO in un'entità JPA, mappando i campi del primo sui corrispondenti attributi del secondo attraverso l'assegnazione diretta dei valori.

Logica di Inserimento del Paziente

Validazione iniziale

Una volta ricevuto il DTO attraverso l'endpoint, si procede con la conversione in entità mediante il mapper. Se il paziente così ottenuto non risulta valido secondo le regole di business definite, si restituisce una risposta con codice 400 Bad Request.

Definizione codice 400 :: Il codice di stato HTTP 400 Bad Request indica che il server non può o non vuole elaborare la richiesta a causa di un errore del client,

come una sintassi non valida o parametri mancanti.

Il corpo della risposta di errore contiene la lista degli errori rilevati durante la validazione, in modo che il client possa comprendere quale dato risulta non conforme alle specifiche.

Controlli di unicità

Successivamente, si effettuano ulteriori controlli per verificare l'unicità di alcuni campi significativi, come l'indirizzo email e il codice fiscale. Tali controlli vengono implementati aggiungendo metodi derivati nel repository.

Definizione metodo derivato :: Un metodo derivato è un metodo del repository Spring Data JPA il cui nome segue una convenzione che permette a Spring di generare automaticamente la query corrispondente, senza necessità di scrivere esplicitamente il codice SQL.

Definizione findByEmail :: `findByEmail` è un metodo derivato che restituisce una lista di pazienti aventi l'indirizzo email specificato come parametro di ricerca, permettendo di verificare l'esistenza di duplicati.

Definizione findBySsn :: `findBySsn` è un metodo derivato che restituisce una lista di pazienti aventi il codice fiscale specificato come parametro di ricerca, permettendo di verificare l'unicità del dato anagrafico.

Gestione errori di duplicazione

Se l'indirizzo email risulta già presente nel database, si costruisce una struttura dati contenente un messaggio descrittivo e si restituisce una risposta Bad Request con il messaggio di errore relativo all'email duplicata.

Analogamente, se il codice fiscale risulta già registrato, si procede con la stessa logica restituendo un messaggio di errore appropriato che segnala il duplicato del codice fiscale.

Salvataggio nel database

Se nessuna di queste condizioni di unicità viene violata, il paziente viene salvato nel database mediante il metodo save del repository.

Definizione save :: `save` è un metodo del repository Spring Data JPA che persiste l'entità nel database, generando se necessario l'identificativo univoco e

restituendo l'entità salvata con tutti i dati aggiornati.

Al termine dell'operazione, si restituisce una risposta OK contenente la versione DTO dell'entità appena salvata, in modo che il client possa visualizzare i dati completi appena inseriti, inclusi eventuali campi generati automaticamente dal database come l'identificativo univoco.

Definizione toDto :: `toDto` è un metodo del mapper che converte un'entità JPA in un DTO, estraendo i campi necessari per la comunicazione con il client e nascondendo eventuali dettagli interni dell'implementazione.

Entità Figlia: Rapporto tra Paziente e Visite

Nel dominio dell'applicazione WebClinic, un paziente può effettuare molte visite mediche. Questa relazione viene modellata come un rapporto uno a molti, dove il paziente rappresenta il lato uno della relazione e la visita rappresenta il lato molti. Dal punto di vista del database, ogni visita contiene un riferimento al paziente a cui appartiene.

Definizione relazione uno a molti :: Una relazione uno a molti, indicata in JPA come `@OneToMany`, stabilisce che un singolo record della tabella padre può essere associato a molteplici record nella tabella figlio, mentre ogni record figlio può avere un solo record padre associato.

Vincoli temporali delle visite

L'applicazione prevede che un paziente possa prenotare al massimo otto visite giornaliere, distribuite in due fasce orarie:

- **Mattina**: dalle ore 9:00 alle 12:00
- **Pomeriggio**: dalle ore 14:00 alle 17:00

Il tempo viene venduto in slot di un'ora ciascuno, il che significa che le visite possono essere prenotate esclusivamente per le ore 9, 10, 11, 12, 14, 15, 16 e 17.

Durante l'inserimento di una nuova visita, risulta necessario verificare che lo slot orario richiesto risulti libero, in modo da evitare sovrapposizioni e garantire che un paziente non abbia più visite programmate nello stesso momento.

Definizione dell'Entità Visit

Per implementare questa logica viene creata una nuova entità denominata Visit, la quale contiene i seguenti attributi:

- Un identificativo univoco di tipo intero
- Una descrizione opzionale di tipo stringa
- Il prezzo della visita di tipo intero
- La data della visita
- L'ora della visita (tipo intero per rappresentare lo slot orario)

Definizione entità Visit :: L'entità Visit modella una singola visita medica effettuata da un paziente, contenendo tutte le informazioni relative alla prenotazione quali la data, l'orario, la descrizione del servizio erogato e il prezzo corrispondente.

Configurazione della Relazione nel Padre

L'entità paziente deve contenere una lista che rappresenta tutte le visite ad essa associate. Questa lista viene dichiarata utilizzando l'annotazione `@OneToMany`, che stabilisce la relazione bidirezionale con l'entità figlio.

Attributo mappedBy

L'annotazione mappedBy indica il nome del campo presente nell'entità figlio che stabilisce il collegamento con il padre, permettendo a JPA di comprendere come le due entità sono correlate attraverso la chiave esterna.

Definizione mappedBy :: L'attributo mappedBy specifica il nome del campo nell'entità proprietaria della relazione, ovvero l'entità figlio, che contiene il riferimento all'entità padre attraverso l'annotazione `@ManyToOne`.

Attributo cascade

L'attributo cascade con valore `CascadeType.ALL` indica che qualsiasi operazione eseguita sull'entità padre, come il salvataggio, l'aggiornamento o la rimozione, deve essere propagata automaticamente a tutte le entità figlio contenute nella lista.

Definizione cascade :: Il meccanismo di cascade permette di propagare le operazioni di persistenza da un'entità padre alle sue entità correlate, evitando la necessità di gestire manualmente ciascun figlio separatamente.

Attributo orphanRemoval

L'attributo orphanRemoval con valore `true` stabilisce che se un elemento viene rimosso dalla lista delle visite, quel record verrà eliminato automaticamente dal database, garantendo la coerenza dei dati.

Definizione orphanRemoval :: L'opzione orphanRemoval, quando impostata a true, causa l'eliminazione automatica dal database di ogni entità figlio che non risulta più referenziata dall'entità padre, eliminando di fatto il legame tra i due record.

Configurazione della Relazione nel Figlio

L'entità Visit deve contenere un riferimento all'entità Patient che rappresenta il padre della relazione. Questo riferimento viene configurato mediante l'annotazione `@ManyToOne`, poiché molte visite possono essere associate a un singolo paziente.

Definizione @ManyToOne :: L'annotazione `@ManyToOne` stabilisce una relazione molti a uno in JPA, indicando che molteplici istanze dell'entità corrente possono essere associate a una singola istanza dell'entità referenziata.

Attributo fetch

L'attributo fetch con valore `FetchType.EAGER` specifica che il paziente associato alla visita deve essere caricato immediatamente dal database insieme alla visita stessa, senza attendere una successiva richiesta esplicita.

Definizione FetchType.EAGER :: Il tipo di caricamento eager indica che i dati correlati vengono recuperati dal database nel momento stesso in cui l'entità principale viene caricata, in contrapposizione al caricamento lazy che posticipa il recupero fino al momento dell'effettivo utilizzo.

Annotazione @JoinColumn

L'annotazione `@JoinColumn` specifica il nome della colonna che verrà creata nel database per memorizzare la chiave esterna. In questo caso, la colonna `patient_id` conterrà il riferimento all'identificativo del paziente appartenente.

Definizione @JoinColumn :: `@JoinColumn` indica il nome della colonna nel database che rappresenta la chiave esterna verso l'entità referenziata, permettendo di

stabilire e gestire il collegamento tra le due tabelle.

Inserimento di una Visita

L'inserimento di una visita risulta più complesso rispetto all'inserimento di un paziente, poiché coinvolge una relazione tra entità. Il DTO relativo alla visita, denominato VisitDTO, non contiene l'intero oggetto paziente ma unicamente il suo identificativo, rappresentato dal campo patientID.

Definizione patientID :: Il patientID è l'identificativo numerico del paziente utilizzato per stabilire il collegamento tra la visita e il paziente a cui appartiene, evitando di dover passare l'intero oggetto Patient nel DTO.

VisitDTOMapper

Per gestire la conversione tra DTO ed entità viene creato un servizio mapper specifico denominato VisitDTOMapper, il quale contiene i metodi per trasformare l'entità in DTO e viceversa.

Definizione VisitDTOMapper :: VisitDTOMapper è una classe di servizio Spring che gestisce la conversione bidirezionale tra l'oggetto VisitDTO e l'entità Visit, occupandosi anche del recupero dell'entità Patient dal database attraverso il repository.

Metodo fromDTO

Il metodo fromDTO esegue le seguenti operazioni in sequenza logica:

1. **Creazione dell'oggetto Visit** → Creazione di un nuovo oggetto Visit vuoto, pronto per essere popolato con i dati provenienti dal DTO.
2. **Copia dei dati semplici** → Copia dell'identificativo, della descrizione, dell'ora e del prezzo dal DTO verso l'entità Visit mediante i rispettivi setter.
3. **Estrazione del patientID** → Il DTO non contiene l'intero oggetto paziente ma esclusivamente il suo identificativo, rappresentato dal campo patientID.
4. **Ricerca del paziente** → Utilizzo del patientID per cercare il paziente nel database attraverso il metodo `findById` del PatientRepository.

5. **Associazione del paziente** → Se il paziente viene trovato nel database, viene associato alla visita mediante il setter appropriato; in caso contrario, viene impostato il valore null per evitare errori di persistenza.
6. **Restituzione della visita** → Restituzione della visita completa, contenente tutti i dati necessari per essere salvata nel database, incluso il collegamento al paziente.

VisitRepository

Il repository per l'entità Visit estende l'interfaccia `JPARepository`, la quale fornisce automaticamente i metodi di base per le operazioni CRUD. Per le esigenze attuali, il repository di base risulta sufficiente, sebbene in futuro potrebbe essere necessario estenderlo con metodi derivati specifici per le ricerche.

Definizione JPARepository :: JPARepository è un'interfaccia di Spring Data JPA che combina le funzionalità di CrudRepository e PagingAndSortingRepository, fornendo metodi standard per le operazioni di persistenza come save, findById, delete e molte altre.

VisitAPI

L'API per la gestione delle visite viene implementata come controller REST utilizzando l'annotazione `@RestController`. L'annotazione `@RequestMapping` posta a livello di classe fornisce un indirizzo di base comune a tutti i metodi del controller, permettendo di organizzare gli endpoint in modo gerarchico e coerente.

Definizione @RequestMapping :: `@RequestMapping` è un'annotazione Spring che definisce il percorso URL di base per tutti i metodi del controller, permettendo di raggruppare gli endpoint correlati sotto uno stesso prefisso.

Annotazione @RequestBody

L'annotazione `@RequestBody` indica che il parametro del metodo deve essere legato al corpo della richiesta HTTP. Grazie a questa annotazione, il JSON contenuto nel corpo della richiesta viene mappato automaticamente sull'oggetto VisitDTO specificato come parametro, senza necessità di parsing manuale.

Definizione @RequestBody :: `@RequestBody` è un'annotazione Spring che segnala che il corpo della richiesta HTTP deve essere deserializzato e convertito

nell'oggetto Java indicato come parametro del metodo, gestendo automaticamente la conversione da formato JSON a oggetto Java.

Con `@RequestBody`, il JSON nel corpo della richiesta viene mappato direttamente su VisitDTO e il metodo può ricevere il DTO come parametro in modo trasparente.

Metodo toDTO

Il metodo toDTO deve essere implementato nel mapper per eseguire la conversione inversa, trasformando un'entità Visit in un VisitDTO. Questo metodo estrae i dati dalla visita e costruisce il DTO corrispondente, includendo l'identificativo del paziente al posto dell'intero oggetto.