ANNOTATIONS

Annotazioni di base

`@Entity` viene utilizzata quando una classe deve essere mappata come tabella del database e gestita da JPA. Ad esempio si applica a classi come `Citizen` o `Sanction` per indicare che rappresentano entità persistenti.

`@Table` si usa quando si vuole specificare esplicitamente il nome della tabella o configurarne alcuni dettagli. Un caso d'uso tipico è quando il nome della tabella nel database è diverso dal nome della classe Java.

`@Id` serve a identificare la chiave primaria dell'entità. Si utilizza quando un attributo deve identificare in modo univoco ogni record della tabella.

`@GeneratedValue` viene usata insieme a `@Id` per indicare che il valore della chiave primaria viene generato automaticamente dal database, ad esempio tramite auto-increment.

`@Column` è utile quando si vuole personalizzare una colonna del database, ad esempio per cambiare il nome della colonna, impostare vincoli come `nullable` o definire la lunghezza di una stringa.

Annotazioni per le relazioni

`@OneToMany` si utilizza quando un'entità padre è collegata a più entità figlie. Un esempio tipico è un cittadino che può avere più sanzioni.

`@ManyToOne` viene usata quando più entità fanno riferimento a una sola entità padre. Nel caso delle sanzioni, ogni sanzione è associata a un solo cittadino.

`@OneToOne` è adatta quando esiste una relazione uno a uno tra due entità, come ad esempio un utente e il suo profilo.

`@ManyToMany` si usa quando più entità possono essere associate a più entità di un altro tipo, come studenti e corsi.

`@JoinColumn` serve a definire la colonna che rappresenta la chiave esterna nel database e permette di collegare due tabelle.

Annotazioni di supporto

`@CascadeType` viene utilizzata all'interno delle annotazioni di relazione per indicare quali operazioni devono propagarsi dall'entità padre a quella figlia. Un caso d'uso comune è `CascadeType.ALL`, che permette di salvare o eliminare automaticamente le entità collegate quando si opera sull'entità principale.
`FetchType` viene utilizzato nelle annotazioni di relazione per definire **quando** i dati associati devono essere caricati dal database.

Esistono due casi d'uso principali.

`FetchType.EAGER` indica che la relazione viene caricata immediatamente insieme all'entità principale. Un esempio tipico è una `@ManyToOne`, come nel caso di `Sanction` verso `Citizen`: quando si carica una sanzione, il cittadino associato viene recuperato subito.

`FetchType.LAZY` indica invece che la relazione viene caricata solo quando viene effettivamente utilizzata. Un caso comune è una `@OneToMany`, come la lista di sanzioni in `Citizen`: il cittadino viene caricato subito, mentre le sanzioni vengono recuperate solo quando si accede alla lista.

L'uso di `FetchType.LAZY` è generalmente preferibile per migliorare le prestazioni ed evitare di caricare dati non necessari, mentre `FetchType.EAGER` è utile quando la relazione è sempre richiesta insieme all'entità principale.

`@RestController` viene utilizzata per definire una classe come controller REST. Indica che la classe è responsabile della gestione delle richieste HTTP e che i metodi restituiscono direttamente dati, solitamente in formato JSON, senza passare da una vista.

`@Autowired` serve per l'iniezione delle dipendenze. Viene usata quando una classe ha bisogno di un'altra componente di Spring, ad esempio un service o un repository, e permette a Spring di fornire automaticamente l'istanza corretta.

`@PostMapping` viene utilizzata per gestire le richieste HTTP di tipo POST. Un caso d'uso tipico è la creazione di una nuova risorsa, ad esempio l'inserimento di un nuovo cittadino o di una nuova sanzione.

`@GetMapping` viene usata per gestire le richieste HTTP di tipo GET. Serve per recuperare dati dal backend, come la lista dei cittadini o il dettaglio di una singola sanzione.

`@PutMapping` viene utilizzata per gestire le richieste HTTP di tipo PUT. È impiegata quando si deve aggiornare una risorsa esistente, ad esempio modificare i dati di

una sanzione già presente nel database.
`@Service` viene utilizzata per indicare una classe che contiene la logica di business dell'applicazione. Serve a separare la gestione delle richieste HTTP, che avviene nei controller, dalle operazioni applicative vere e proprie, come controlli, calcoli o coordinamento tra repository.

Una classe annotata con `@Service` viene gestita da Spring come componente e può essere iniettata tramite `@Autowired` all'interno dei controller o di altri service. Un caso d'uso tipico è un servizio che si occupa di creare, aggiornare o recuperare entità dal database applicando eventuali regole di business prima di delegare le operazioni ai repository.