**GIORNO 03**

**CRUD in Spring Boot**

Repo del giorno: https://github.com/nclBaz/epicode-it-u5-d8-2

Gestione degli errori

Lato Back End si parla di “uno per tutti” nel senso che, se è vero che lato FE abbiamo un applicativo che gira per ogni browser per ogni Client e ogni eccezione sarà, in genere, “problema” del Client. Lato BE avremo, in generale, un solo Server. È quindi di fondamentale importanza evitare che possa avere problemi e che possa crashare. Questo si traduce dicendo che gli errori 400 sono, per quanto gravi, meno dannosi rispetto agli errori 500. Si utilizzano una piccola porzione di tutti gli StatusCode codificati perché molti sono estremamente selettivi e specifici. Errore 400: Rappresenta un errore causato dalla richiesta del client. Ad esempio, richieste malformate o parametri mancanti.

Errore 500: Rappresenta un errore interno del server. Ad esempio, problemi di connessione al database o errori nel codice lato server.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteL’obiettivo è quello di avere un gestore delle eccezioni globale, ovvero un'unica Classe che riceve un errore inviando la risposta corretta specifica per quell’errore. Tale Classe è annotata con @RestControllerAdvice e avrà tutta una serie di metodi che raccolgono le eccezioni e, in base a come sono definite, ritornano una specifica risposta.

Immagine che contiene testo, Software multimediale, software, Carattere

Descrizione generata automaticamenteCreiamo una Classe NotFoundException che estende Exception. (L’idea è quella di creare una serie di Classi per rappresentare tutte le eccezioni che possiamo incontrare (Not Found, Bed Request, ecc…).

Create le Classi ExceptionCustom possiamo lanciarle in tutti i metodi che avranno bisogno di gestire quella specifica eccezione. Nel caso della Classe NotFoundException allora la utilizziamo nei metodi GET da id:

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamenteUna volta definita l’eccezione custom e “lanciata” nei giusti endpoint, allora in EcxeptionHandles specifichiamo i metodi che devono gestire le eccezioni custom tramite @ExceptionHandler(NomeEccezioneCustom.class) e passo nel metodo stesso l’eccezione custom:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Ora in Postman se facciamo una GET per un id che non esiste, mi ritorna il messaggio NOT FOUND.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamenteSe volessimo anche cambiare lo StatusCode di ritorno (esempio portarlo a 404 in caso di Not Found), allora annotiamo ogni metodo con @ResponseStatus che descriva lo Stato specifico di ritorno (404,400,500,recc…)

Siccome non vogliamo ritornarci stringhe, allora creiamo una Classe ErrorsPayload con cui definire meglio il messaggio di ritorno dei metodi (internalCode è il numero aziendale):

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente….e in PostMan, se faccio una ricerca con id inesistente:

Immagine che contiene testo, Software multimediale, software, Software per la grafica

Descrizione generata automaticamenteLa comodità di questa gestione è che se un giorno volessi modificare il tipo di ritorno del messaggio, ad esempio aggiungere un campo in ErrorsPayload “descrizione”, allora mi basterà aggiungere un attributo String description in ErrorsPayloads e inserisco la descrizione direttamente nel return di ogni metodo del ExceptionHandler:

In questo modo evitiamo anche i muri di testo che descrivono le eccezioni.

La cosa migliore da fare, in realtà, sarebbe:

Immagine che contiene testo, software, Software multimediale, Software per la grafica

Descrizione generata automaticamente

cioè mettere un log che faccia comparire in console l’errore specifico consultabile dallo sviluppatore, (log.error(e.getMessage()) e un messaggio di errore generico nel costruttore dell ErrorsPayload(nell’esempio “Errore brutto brutto”). Sarà questo messaggio a comparire in Postman e quindi visibile al Client. Il log.error è fondamentale per il debug.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

L’application.properties contiene informazioni sensibili. Per evitare di renderli pibblici col commit allora:

1. Creo un file .txt esterno al progetto(nell’esempio è env.properties).
2. Inserisco tutti i valori sensibili e segreti:
3. SECRET = mysupersecret
4. PG\_USERNAME = postgres

.

.

.

1. Inserisco il file env.properties con i segreti nel file .gitignore in modo che non venga committato.

Nell’application.properties per farmi ritornare tutti quei valori che abbiamo secretato:

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

I valori ${valore} nell’App.Prop sono quelli che abbiamo definito nel file .txt

**REST Api con Spring Boot – CRUD**

Creiamo un nuovo Progetto:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

…e lo importiamo.

Configuriamo l’application.properties tramite l’env.properties e inseriamo quest’ultimo nel .gitignore.

Immagine che contiene testo, software, Software multimediale, Icona del computer

Descrizione generata automaticamente

Creiamo il database e avviamo.

Creo una Classe User, una NewUserResponsePayload e una UserRequestPayload:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Le ultime due Classi ci serviranno per definire meglio i metodi (e quindi gli endpoints) nella Classe Controller.

Quando andremo a fare una Post per inserire un nuovo User in {baseUrl}/users, noi come body inviamo un oggetto che non ha un id (perché sarà il DB a crearlo) e ciò che ci ritornerà sarà un qualcosa di tipo NewUserResponsePaylod che, di fatto, ha come attributo solamente un id. Discorso analogo per il @PutMapping: noi vogliamo modificare un body che è un UserRequestPayload (name,surmane,email) senza avere la possibilità di modificarne anche l’id, che, anzi, sarà la chiave per “pescare” quel particolare User. Una volta pescato, grazie all’UUID id nel metodo, e aver detto cosa modificare, grazie al body di tipoUserRequestPayload, ci verrà restituito un User con id, name, surname, email.

Definiamo la struttura della Classe Controller con @RestController e @RequestMapping(“/users”) e 5 endpoints:

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, software, Software multimediale

Descrizione generata automaticamente

Creiamo una UserRepository che extends JpaRepository:

Creiamo UserService che mi serve per aggiungere della logica business prima di salvare nel DB tramite la UserRepository:

Immagine che contiene testo, schermata, software, Carattere

Descrizione generata automaticamente

Possiamo definire usersRepo come final (cioè come una costante), purchè il suo valore venga inizializzato in un costruttore altrimenti Eclipse mi da errore (in particolare Eclipse mi costringerebbe a inizializzarla).

In alternativa avremmo potuto anche fare:

@Autowired

Private UsersRepository usersRepo

senza la necessità di definire un costruttore per la Classe Service. Entrambe le strategie sono valide.

Dobbiamo definire ora i metodi di UserService con possibili controlli.

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

L’eccezione NotFoundException è quella creata precedentemente:

Immagine che contiene testo, Software multimediale, software, Carattere

Descrizione generata automaticamente

Una volta definito UserService, possiamo ora definire meglio i metodi dichiarati in UserControll importando con @Autowired UserService:

Immagine che contiene testo, software, Software multimediale, schermata

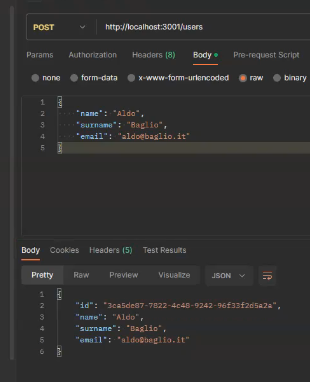
Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, multimediale

Descrizione generata automaticamente

NB. Il metodo Post in PostMan dava un errore che Riccardo ha sistemato modificando il metodo create nell UserService. Da VEDERE!

In PostMan possiamo testare il PostMapping dando come Body un UserRequestPayload (name,surname,email)



e nel DB:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, documento, ricevuta

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Nelle Repo possiamo definire metodi custom che vengono “attivati” in base al nome che diamo.

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

…ed eventualmente possiamo scrivere una Query custom.

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Esempio di Query: controllo se una email è già utilizzata.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteNell’UsersRepository definiamo un metodo findByEmail che ci restituisce un Optional.

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#appendix.query.method.subject>

Creo una BedRequestException:

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente…che lanceremo nel metodo create in UserService

Immagine che contiene testo, Software multimediale, Carattere, software

Descrizione generata automaticamenteIn UsersService definiamo meglio il metodo create

E in ExceptionHandler:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**PAGINAZIONE**

La paginazione nel backend è una tecnica utilizzata per suddividere grandi set di dati in pagine più piccole al fine di migliorare le prestazioni e l'esperienza utente. Consente di recuperare solo una porzione dei dati alla volta, riducendo il carico sul server e riducendo i tempi di risposta. La paginazione è spesso implementata attraverso i parametri di query HTTP come "pagina" e "dimensione", permettendo ai client di specificare quale pagina desiderano visualizzare e quante voci per pagina. Questo è comune nelle API RESTful e nei risultati di query del database. Un esempio è la paginazione in Google e l’infinity scroll.

Noi utilizziamo la paginazione quando ci viene ritornata una List (User, Product, Book, ecc…)

Immagine che contiene testo, Carattere, linea, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, documento, schermata

Descrizione generata automaticamente

In UserService invece di tornarci una List<User>, ci facciamo tornare Page<User> in find():



…e nel UserController:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

In PostMan:

Immagine che contiene testo, software, Software multimediale, Software per la grafica

Descrizione generata automaticamenteAvremo informazioni sul numero totale di pagine, il numero totale di elementi, e il numero della pagina: “number”:1.

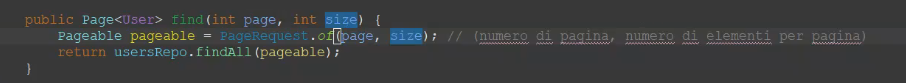
Per rendere dinamica la paginazione, allora in find() di UserService:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente…e in UserController:

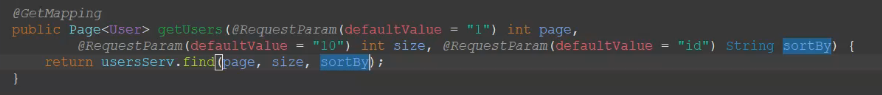
Immagine che contiene testo, linea, Carattere, schermata

Descrizione generata automaticamenteOra posso rendere effettivamente dinamica la paginazione con i parametri di size e page nell’url.

Possiamo passare anche il criterio di Sorting:

Immagine che contiene testo, schermata, Software multimediale, Carattere

Descrizione generata automaticamenteUserService:

UserController:

defaultValue: se non mi passi la pagina tra i Params, allora mi prendo il valore di default.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

…ora possiamo fare ordinare gli elementi in base ad un valore che vogliamo.