illimity	**Redatto da:** Valerio Bondì
	29 Giugno 2022
Oggetto	**Stato**
Documentazione progetto IllimityCodeChallenge	Final Draft

Sommario

Introduzione

Il progetto IllimityCodeChallenge consiste in una semplice applicazione Spring Boot che si occupa di gestire dei **clienti** e dei **movimenti** secondo i problemi definiti nel dataset dato in consegna alla challenge.

I clienti e i movimenti verranno modellati da due classi di persistenza, rispettivamente **Customer** e **Movement**.

Per ognuno dei 2 modelli vengono forniti, oltre i metodi riguardanti la risoluzione della challenge, dei metodi per la lettura, scrittura ed eliminazione di un customer o movement a piacimento.

Viene inoltre fornito un metodo **init**, richiamabile da API, che consente di inizializzare il database con dei customers e dei movements per eseguire le richieste della challenge senza doverli inserire manualmente.

Le tecnologie utilizzate nel progetto sono:

- Java 11;
- Spring 2.7.1;
- MongoDB 5.0.9;

Su sistema operativo Windows 10 Pro e con IDE Intellij.

Setup

Per effettuare il setup del progetto sono stati eseguiti i seguenti passaggi:

- Scaricare MongoDB ed installarlo con le impostazioni di default: ciò comporta nessuna utenza per effettuare la connessione al db e l'indirizzo dell'istanza attiva su localhost:27017
- Creare un progetto Spring Initializr selezionando come dipendenze Spring Web e Spring Data MongoDB.

Una volta creato il progetto, si può definire nell'**application.properties** il nome del database MongoDB da utilizzare (in questo caso "illimity") e la porta di ascolto del server Tomcat.

Struttura

L'applicazione possiede una struttura organizzata mediante i seguenti package:

- **Controller:** i controller dell'applicazione, dove sono definite le varie API REST di interfaccia verso l'esterno;
- **Converter:** classi che consentono di convertire i dati in ingresso e in uscita, in modo tale da non avere un singolo modello per tutto;
- **Exception:** le eccezioni e gli handler custom dell'applicazione;
- **Model:** le classi che modellano i dati del dominio dell'applicazione;
- **Repository:** interfacce di comunicazione per le operazioni verso il database;
- **Response:** contiene la classe ErrorResponse per la gestione degli errori in risposta alle API
- **Service:** contiene le classi che modellano la logica di business per i customer e i movement.

Business logic

La logica di business è suddivisa in 3 classi:

CustomerService

Questa classe fornisce dei metodi per lettura, scrittura ed eliminazione dei vari customers. Queste operazioni non hanno altra logica complessa, a meno del *createCustomer* dove inizialmente viene fatto un controllo che non si possa inserire più un Customer con lo stesso username.

Inoltre c'è il metodo *login(String username, String password)* che consente di effettuare il login di un customer. Vengono controllate 3 condizioni:

- Se il customer della quale si vuole fare login esiste;
- Se la password inserita matcha con quella salvata;
- Se lo status del customer è ACTIVE.

Se le 3 condizioni sono soddisfatte, allora il login viene effettuato con successo e viene restituito lo status 200 in risposta all'API con alcuni dati del customer. Altrimenti viene restituita un'eccezione descrittiva del problema incontrato (è stato differenziato l'errore fra utente inesistente e password errata, ma potrebbe essere anche una buona scelta unificare entrambi i casi in 401 UNAUTHORIZED)

MovementService

Questa classe fornisce dei metodi per lettura, scrittura ed eliminazione dei vari customers. Queste operazioni non hanno altra logica complessa.

Inoltre c'è il metodo *findAllMovementsByCustomerId()* che si preoccupa di ottenere la lista di tutti i movement per un determinato customerId.

Il metodo è paginato, ovvero accetta un oggetto **Pageable** che consente di effettuare paginazione ed ordinamento dei risultati.

Come scelta progettuale, si è deciso di lasciare "libera" la paginazione. Quindi per ottenere il secondo quesito della challenge, è sufficiente richiamare l'API per ottenere i movimenti paginati con dei query params che descrivano un risultato di 10 elementi ordinati per data decrescente.

A seguito un esempio per un customerId generico:

http://localhost:8090/movements/paginated/67bca135-c1ec-47c7-9e6d-0e9bd9e2b132?page=0&size=10&sort=date,DESC

NB: come ID dei Customer e Movement è stato scelto di utilizzare un oggetto UUID che consente di avere un id univoco.

PasswordEncoder

Effettua l'encoding di una stringa e espone un metodo *matches(String rawPassword, String encodedPassword)* che controlla se l'encoding della rawPassword in input equivale all'encodedPassword.

Siccome da consegna della challenge non erano richiesta l'implementazione dei metodi, essi richiamano semplicemente l'implementazione della comune libreria di Apache Commons Codec.

Gestione delle eccezioni

Le eccezioni custom dell'applicazione vengono gestite mediante la classe **ResponseException**. Tale classe estende RuntimeException e ha come attributo un oggetto di tipo **ResponseErrorEnum** dove sono definiti degli enum aventi:

- Un codice numerico identificativo dell'errore;
- Un messaggio di errore;
- Lo status http dell'errore.

Tali eccezioni vengono gestite in uscita mediante un **handler** globale definito dalla classe **GlobalControllerExceptionHandler.** Tale classe cattura:

- Le eccezioni di validazione dei bean in ingresso (ad es. se viene violata una validazione *@NotNull*);
- Le eccezioni di parsing (ad es. se si passa ad un enum un valore non atteso)
- Le eccezioni ResponseException.

Tutte e 3 i tipi di eccezioni vengono rimappate in una classe **ErrorResponse** che fornisce:

- Il timestamp dell'eccezione;
- Il codice http dell'errore;
- Il messaggio di errore.

Test

Anche se non richiesti dal problema, il progetto è dotato di **Unit Test** mediante JUnit 5 per testare a livello unitario le varie classi (Controller compresi gli handler delle eccezioni, Converter e Service) e **Mockito** per la gestione dei mock, consentendo così una code coverage dell'83%.

Non vi sono degli **Integration Test** automatizzati, ma sono stati effettuati dei test di integrazione creando una **Collection Postman** di API utili per creare, leggere, eliminare customer e movement e per il login e la get paginata dei movimenti (verificando anche i casi di errore)

Documentazione OpenAPI

Tramite l'importazione della dipendenza OpenAPI 3.0, è possibile visionare la struttura delle API all'indirizzo (una volta avviata l'applicazione):

http://localhost:8090/swagger-ui/index.html