Applicazioni Internet 2019-20

Esercitazione 2

Un'applicazione web permette di gestire la creazione di gruppi di studenti all'interno di corsi universitari.

1. Si crei un progetto Spring Initializr includendo i moduli Lombok, Spring Web, Thymeleaf, Spring Data JPA, MySQL driver

Le informazioni gestite sono salvate in un DBMS relazionale.

2. Si utilizzi la versione basata su Docker di MariaDB: si scarichi l'immagine dal sito https://hub.docker.com/_/mariadb e si seguano le indicazioni contenute nella pagina per attivare un container, avendo cura di esportare la porta 3306.
   Dal menu Database di IntelliJ Idea, creare una nuova sorgente dati (datasource) indicando come DBMS MySQL, utilizzando come utente 'root' e password quella assegnata all'atto dell'avvio del contenitore. Si lasci vuoto il campo "database". Usando il bottone "Test Connection" si verifichi la correttezza delle impostazioni.
   Si selezioni la sorgente dati create e si prema il tasto "Source Editor" per attivare la console. Al suo interno di digiti il comando "CREATE DATABASE teams;" e lo si esegua, verificando che non si generino errori.

3. Si crei una nuova sorgente dati, indicando le stesse credenziali di prima, ma assegnando al campo "database" il valore "teams". Si verifichi che la connessione funziona correttamente.
   Si interrompa il container del dbms (docker container stop <containerId>), lo si elimini (docker container rm <containerId>) e lo si ricrei con lo stesso comando usato in precedenza. Tornando sulla console di IntelliJ si verifichi che la prima sorgente dati (@localhost) continua a funzionare, mentre la seconda (teams@localhost) non funziona più: perché succede questo comportamento?

4. Si interrompa e rimuova nuovamente il container e lo si ricrei aggiungendo l'indicazione di un volume (si legga attentamente il contenuto del paragrafo "Where to Store Data" della pagina di DockerHub).
   Si esegua nuovamente il comando "CREATE DATABASE teams;" si controlli che la seconda sorgente dati funziona correttamente, si fermi, elimini e ricrei il container e si verifichi che questa volta i dati salvati non sono andati persi con l'eliminazione del container.

   ATTENZIONE: la cartella usata come base del volume contiene i file usati da MariaDB: se questi vengono manipolati/cancellati nel sistema host, il contenitore non sarà in grado di funzionare. Inoltre, il contenitore non potrà essere spostato in modo indipendente su un'altra macchina: tutti i dati sono infatti contenuti nella cartella montata come volume.

Un docente ha la possibilità di

- aggiungere un nuovo corso specificando la dimensione minima e massima dei gruppi al suo interno

5. Nel progetto SpringBoot, si creino i package "services", "dtos", "repositories", "entities" e "controllers".
   Si aggiunga, nel package entities, la classe "Course" etichettandola come @Entity (per renderla oggetto di persistenza) e @Data (per generare getter/setter). Al suo interno si creino i campi "name" (di tipo String, etichettato con @Id), "min" e "max" (di tipo int).
   Si aggiungano, nel file "application.properties", le informazioni necessarie a collegarsi alla base dati,

ricavando i valori dal pannello dei db (spring.datasource.url, spring.datasource.username, spring.datasource.password, spring.datasource.driver-class-name).
Si aggiungano, inoltre, le chiavi spring.jpa.generate-ddl e spring.jpa.show-sql, impostandole a true.
Nel package repositories si aggiunga l'interfaccia CourseRepository che estende JpaRepository<Course,String>, annotandola con @Repository.
Si esegua il progetto verificando, nei log e nel pannello dei database, che venga correttamente create una nuova tabella chiamata "course" con tre colonne al proprio interno. Si fermi l'applicazione.

6. Usando il pannello dei database, inserire nella tabella creata alcune righe e salvarle nel DBMS.
Si aggiunga, nella classe principale dell'applicazione un metodo etichettato con @Bean che accetta come parametro un oggetto di tipo CourseRepository. Tale metodo deve ritornare un'istanza dell'interfaccia CommandLineRunner. Nel corpo del metodo run() di tale interfaccia si utilizzi il repostory iniettato per stampare a video tutti gli oggetti di tipo Course.
Si esegua il progetto e verifichi che tutte le righe aggiunte manualmente sono effettivamente mostrate.

- abilitare o disabilitare un corso (impedendo così ulteriori evoluzioni del suo stato)

7. Si aggiunga, nella classe Course, il campo "enabled" di tipo boolean. Si rilanci il progetto e si verifichi che è presente una nuova colonna nella tabella corrispondente e che i dati precedenti non sono andati perduti.

- aggiungere singolarmente studenti ad un corso

8. Si aggiunga, nel package entities, la classe Student, con i campi "id" (@Id String), "name" (String) e "firstName" (String) e la si annoti con @Data e @Entity
Si aggiunga, nel package repositories, l'interfaccia StudentRepository che estende JpaRepository<Student, String> e la si annoti con @Repository.

9. Si modifichi il CommandLineRunner del metodo principale perché elenchi gli studenti presenti nel DBMS, si aggiungano manualmente alcune righe e si verifichi che tutto funziona regolarmente.
Si crei la relazione molti-a-molti tra corsi e studenti: nella classe Student si aggiunga il campo "courses" di tipo List<Course> e lo si annoti con @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}): tale annotazione informa il sistema di persistenza che eventuali modifiche di elementi alla lista dovranno essere propagate sul database. Per permettere al DBMS di tenere traccia di questa informazione, si aggiunga a tale campo anche l'annotazione @JoinTable(name="student_course", joinColumns = @JoinColumn(name="student_id"), inverseJoinColumns = @JoinColumn(name="course_name")): tale annotazione indica che occorre creare una nuova tabella (student_course) che contiene due colonne (student_id e course_name) che sono chiavi esterne delle tabelle student e course. Il contenuto di questa tabella verrà utilizzato per valorizzare i dati presenti nella lista.
Nella classe Course, si aggiunga il campo "students", di tipo List<Student>, etichettato con @ManyToMany(mappedBy = "courses"): tale annotazione indica che i dettagli per valorizzare tale relazione sono da ricavare dal lato opposto della relazione, ovvero da quanto riportato nelle annotazioni del campo "courses" della classe Student). Si inizializzino entrambe le liste con una nuova istanza di ArrayList<>().
In entrambe le classi si aggiunga un metodo opportuno (addStudent(…) o addCourse(…)) che provvede ad aggiungere nella propria lista di elementi correlati il parametro ricevuto e nella lista duale contenuta nel parametro l'elemento this, così da garantire la simmetricità dell'informazione (se S.getCourses() contiene C, allora C.getStudents() deve includere S).

10. Si esegua l'applicazione e si verifichi sia che il DB viene modificato correttamente, sia che l'applicazione lancia un'eccezione che ne comporta l'arresto (java.lang.IllegalStateException: Failed

to execute CommandLineRunner).
Questo capita perché, per motivi di ottimizzazione, il sistema di persistenza evita di popolare direttamente i campi di tipo relazione, sostituendo il loro contenuto con dei proxy, che potranno essere valorizzati qualora si tentasse di fare accesso al loro contenuto. Tale comportamento, tuttavia, funziona correttamente solo se ci si trova nel contesto di una transazione (cosa che non succede nel CommandLineRunner).
Nel package dtos si creino le classi StudentDTO e CourseDTO, etichettandole con @Data e si aggiungano, al loro interno, gli stessi campi delle classi Student e Course, tranne le due liste.
Nel package services, si crei l'interfaccia TeamService composta dai seguenti metodi:
- boolean addCourse(CourseDTO course);
- Optional<CourseDTO> getCourse(String name)
- List<CourseDTO> getAllCourses();
- boolean addStudent(StudentDTO student);
- Optional<StudentDTO> getStudent(String studentId);
- List<StudentDTO> getAllStudents();
- List<StudentDTO> getEnrolledStudents(String courseName);
- boolean addStudentToCourse(String studentId, String courseName);
- void enableCourse(String courseName);
- void disableCourse(String courseName);

11. Si aggiunga alle dipendenze del progetto la libreria org.modelmapper:modelmapper:2.3.6 e si inserisca, nella classe principale dell'applicazione, un bean di tipo ModelMapper.
Nello package services, si aggiunga la classe TeamServiceImpl che implementa l'interfaccia corrispodente e la si annoti con @Service e con @Transactional.
Al suo interno potranno essere iniettati i repository necessari per realizzare la logica richiesta oltre al ModelMapper. I metodi di inserimento restituiscono true se il dato passato non esisteva ancora, false altrimenti.
I metodi getEnrolledStudents(…), addStudentToCourse(…), enableCourse(…) e disableCourse(…) devono lanciare un'eccezione se la chiave primaria fornita non esiste. Si creino a tale scopo due apposite classi (StudentNotFoundException e CourseNotFoundException) che estendono TeamServiceException che a sua volta estende RuntimeException.
Si modifichi il CommandLineRunner perché operi su un'istanza di TeamService e si verifichi che le operazioni di ricerca, inserimento e elencazione funzionano correttamente.

- aggiungere l'elenco degli studenti iscritti ad un corso a partire da un file CSV

12. Si aggiungano, nell'interfaccia TeamService, i metodi
- List<Boolean> addAll(List<StudentDTO> students);
- List<Boolean> enrollAll(List<String> studentIds, String courseName);
e li si implementino nella classe TeamServiceImpl.
Si legga attentamente il documento https://attacomsian.com/blog/spring-boot-upload-parse-csv-file# e si aggiunga nell'interfaccia TeamService il metodo
- List<Boolean> addAndEroll(Reader r, String courseName)
Si provveda quindi ad implementarlo secondo le indicazioni del documento citato.
Si verifichi che tale metodo permette di importare correttamente un file prelevato dal filesystem locale

Uno studente può:

- Vedere l'elenco dei corsi a cui è iscritto

13. Nell'interfaccia TeamService si aggiunga il metodo
    - List<CourseDTO> getCourses(String studentId);
    e lo si implementi.

- Vedere l'elenco dei gruppi a cui è iscritto

14. Nel package entities, si crei la classe Team, etichettandola con @Entity e @Data. Al suo interno si aggiunga il campo 'id' di tipo Long (fate attenzione che sia scritto con l'iniziale maiuscola), etichettandolo con @Id e @GeneratedValue, i campi 'name' di tipo String e 'status' di tipo int. Si aggiunga inoltre il campo 'course' di tipo Course e lo si etichetti con @ManyToOne e @JoinColumn(name= "course_id"): questo campo rappresenta infatti il corso cui questo gruppo appartiene.
    Nella classe Course, si aggiunga un campo di tipo List<Team> chiamato 'Teams' e lo si etichetti con @OneToMany(mappedBy = "course").
    In entrambe le classi si aggiungano metodi per aggiungere e togliere in modo simmetrico elementi alla collezione (in particolare, nella classe Team, si implementi il metodo setCourse(…) distinguendo se il nuovo valore è o meno nullo, per decidere se togliere o aggiungere il gruppo dalla lista del corso).
    Si aggiunga la classe TeamRepository nel package corrispondente, sulla falsa riga di quanto fatto in precedenza.

15. Nella classe Team, si aggiunga un campo di tipo List<Student> chiamato 'members', etichettato con @ManyToMany(cascade = …) e @JoinTable(…), inizializzato con un ArrayList nuovo
    Nella classe Student, si aggiunga un campo di tipo List<Team> chiamato 'teams' etichiettato con @ManyToMany(mappedBy = "members"), anch'esso inizializzato opportunamente.
    In entrambe le classi, si inserisca una coppia di metodi di supporto per l'inserimento e la rimozione simmetrica sui due lati di un elemento dalla lista.
    Nel package dtos, si aggiunga la classe TeamDTO inserendo i campi elementari corrispondenti.
    Si modifichi l'interfaccia di TeamService aggiungendo i seguenti metodi
    - List<TeamDTO> getTeamsForStudent(String studentId);
    - List<StudentDTO>getMembers(Long TeamId);
    e li si implementi. Si verifichi che la funzionalità introdotta non rompa nessuno dei meccanismi in essere.

- Proporre, con altri studenti, la formazione di un gruppo in un dato corso, a condizione che il corso sia abilitato, tutti i membri proposti risultino iscritti al corso, non facciano già parte di altri gruppi nell'ambito dello stesso corso, che siano rispettati i vincoli di cardinalità definiti nell'ambito del corso e non vi siano duplicati nell'elenco dei partecipanti

16. Si aggiunga, nell'interfaccia TeamService, il seguente metodo
    - TeamDTO proposeTeam(String courseId, String name, List<String> memberIds);
    e lo si implementi, rispettando tutti i vincoli indicati. Qualsiasi violazione deve comportare il lancio di un'opportuna eccezione.
    Si faccia attenzione al fatto che l'identificativo del gruppo è assegnato automaticamente all'atto della richiesta di salvataggio: a tale scopo il metodo .save(…) del repository restituisce un'istanza della stessa classe salvata, con il campo 'id' correttamente valorizzato.
    Si modifichi il CommandLineRunner per verificare li corretto funzionamento del metodo.

- Vedere l'elenco dei gruppi di un dato corso

17. Si aggiunga, nell'interfaccia TeamService, il metodo
    - List<TeamDTO> getTeamForCourse(String courseName);
    e la si implementi

- Vedere la lista di studenti di un corso che partecipano già o che non partecipano ancora ad un gruppo

18. Si aggiunga, nell'interfaccia CourseRepository, il seguente metodo
    - List<Student> getStudentsInTeams(String courseName);
    e lo si annoti con l'espressione
    `@Query("SELECT s FROM Student s INNER JOIN s.teams t INNER JOIN t.course c WHERE c.name=:courseName")`.
    Tale espressione è un esempio di interrogazione JPQL. Essa permette di descrivere, in termini di entità e non di linguaggio SQL, interrogazioni specifiche. Nel caso specifico, essa consente di navigare seguendo le relazioni in essere dall'insieme di tutti gli studenti a soli quelli che partecipano ad almeno un gruppo che appartenga al corso specificato dal parametro.
    Si aggiunga, nell'interfaccia del servizio il metodo
    - List<StudentDTO> getStudentsInTeams(String courseName);
    e lo si implementi appoggiandosi all'estensione introdotta nel repository.
    Si verifichi che funziona correttamente.
    Esaminando i log dell'applicazione, si determini l'interrogazione SQL effettivamente generata e se ne valuti la complessità.

19. Si aggiunga, nell'interfaccia CourseRepository, il metodo ulteriore
    - List<Student> getStudentsNotInTeams(String courseName);
    e si determini quale annotazione assegnargli, in modo tale che restituisca l'insieme differenza tra tutti gli iscritti al corso e quelli che già partecipano ad un gruppo.
    Si aggiunga nell'interfaccia del servizio il metodo
    - List<StudentDTO> getAvailableStudents(String courseName);
    lo si implementi e se ne verifichi il corretto funzionamento.
    Esaminando i log dell'applicazione, si determini l'interrogazione SQL effettivamente generata e se ne valuti la complessità.

Si realizzi **lo strato di servizio** di tale sistema utilizzando il framework SpringBoot.

Riferimenti

- Uploading and Parsing CSV File using Spring Boot - https://attacomsian.com/blog/spring-boot-upload-parse-csv-file#
- Many-To-Many Relationship in JPA - https://www.baeldung.com/jpa-many-to-many
- Difference between @JoinColumn and mappedBy https://www.baeldung.com/jpa-joincolumn-vs-mappedby
- JPA join types https://www.baeldung.com/jpa-join-types
- Java Persistence/JPQL https://en.wikibooks.org/wiki/Java_Persistence/JPQL