Applicazioni Internet 2019-20

Esercitazione 3

Un'applicazione web permette di gestire la creazione di gruppi di studenti all'interno di corsi universitari. Essa è basata su una API REST che viene comandata da un client realizzato in htm/css/javascript sotto forma di SinglePageApplication.

1. Si estenda il progetto dell'esercitazione precedente. Se si sta usando un sistema di controllo delle versioni, si crei un nuovo ramo, denominato lab3. Se ancora presente, si elimini dalla classe principale il bean di tipo CommandRunner e si esegua un commit.
   Si importi il package spring-boot-starter-hateoas e si sincronizzino le dipendenze di Maven.
   Se non ancora presente, si installi il tool HTTPie (https://httpie.org) disponibile per tutti i sistemi operativi: esso permette di effettuare in modo semplice richieste http dalla riga di comando.

Il servizio offre i seguenti endpoint:

- GET /API/courses – elenca tutti i corsi gestiti dalla piattaforma
- GET /API/courses/{name} – fornisce i dettagli di un singolo corso

2. Nel package controllers, si aggiunga una classe denominata CourseController, la si etichetti con @RestController e con @RequestMapping("/API/courses").
   All'interno di tale classe si inietti un'istanza di TeamService.
   Si introduca il metodo all() che restituisce una lista di CourseDTO. Tale metodo è annotato con @GetMapping({"", "/"}).
   Si implementi il metodo e si verifichi, eseguendo da terminale il comando `http http://localhost:8080/API/courses`, che venga restituita correttamente la lista dei corsi presenti nel db.
   Si aggiunga il metodo getOne(String name) che restituiscce un CourseDTO e lo si annoti con @GetMapping("/{name}").
   Si annoti il parametro del metodo con @PathVariabile per poter accedere correttamente a quanto contenuto nella URL. Si implementi tale metodo attraverso la funzionalità corrispondete offerta dal TeamService. Nel caso in cui il corso non venisse trovato, si lanci un'ecccezione di tipo ResponseStatusException(HttpStatus.CONFLICT, {name}).
   Si verifichi che il controllore permette correttamente di accedere ai dettagli di un singolo corso.

3. Si modifichi la classe CourseDTO facendole estendere RepresentationalModel<CourseDTO>
   Questo permette di arricchire le istanze di questa classe con link ipertestuali, coerentemente con il modello HateOAS.
   Si crei nel package controllers la classe ModelHelper che contiene il metodo pubblico statico enrich(…) che riceve e restituisce un CourseDTO. Seguendo le indicazioni contenute nel documento https://www.baeldung.com/spring-hateoas-tutorial, si aggiunga all'oggetto ricevuto un link a se stesso, e si restituisca l'oggetto così modificato.
   Nei metodi all() e getOne(…), si provveda, prima di restituire i risultati, ad invocare per ciascun CourseDTO trattato, il metodo ModelHelper. enrich(…).
   Si verifichi che gli oggetti restituiti dalle URL implementate contengono correttamente un link a se stessi.

- GET /API/students – restituisce la lista di tutti gli studenti presenti nella piattaforma

- GET /API/students/{id} – fornisce i dettagli di un singolo studente.

4. Si aggiunga la classe StudentController sulla falsa riga di quanto effettuato nel punto precedente, mappando correttamente le URL indicate.
   Si modifichi la classe StudentDTO affinché erediti da RepresentationalModel<StudentDTO>
   Nella classe ModelHelper, si introduca una versione overloaded del metodo enrich(…) che riceva e restituisca un oggetto di tipo StudentDTO a cui aggiunge opportuno link a se stesso.
   Si verifichi che gli oggetti restituiti dalle URL implementate siano corretti e contengano un link a se stessi.

- GET /API/courses/{name}/enrolled – lista degli studenti iscritti ad un dato corso

5. Si aggiunga, nella classe CourseController, il metodo List<StudentDTO> enrolledStudents(String name) che si occupa di restituire la lista degli iscritti ad un corso, annotandolo debitamente.
   Si modifichi il metodo enrich(CourseDTO) della classe ModelHelper per aggiungere ai modelli dei singoli corsi anche la relazione "enrolled" che punta alla URL corrispondente.
   Si verifichi il corretto funzionamento dell'end-point.

- POST /API/courses – aggiunge alla lista dei corsi un nuovo elemento di tipo CourseDTO

6. Nella classe CourseController si aggiunga il metodo addCourse(CourseDTO dto), annotato con @PostMapping({"","/"}). Si annoti il parametro con @RequestBody. Tale metodo dovrà provare ad inserire il dto ricevuto nel sistema servendosi dell'opportuno metodo del servizio. In caso positivo, restituisce la copia dell'oggetto in ingresso arricchita con le relazioni tipiche dei corsi. In caso di fallimento, si lanci un'eccezione di tipo ResponseStatusException(HttpStatus.CONFLICT, {dot.name}).
   Si verifichi, mediante il comando http POST …, che un primo inserimento va a buon fine, mentre tentativi successivi di inserire lo stesso corso falliscono con errore 409.

- POST /API/students – aggiunge alla lista degli studenti un nuovo elemento di tipo StudentDTO
- POST /API/courses/{name}/enrollOne – aggiunge alla lista degli iscritti lo studente indicato dal relativo ID
- POST /API/courses/{name}/enrollMany – accetta un oggetto di tipo multipart/form-data contenente un file di tipo text/csv indicante una lista di studenti che vengono iscritti alla piattaforma e al corso e restituisce una lista di valori boolean

7. Si prosegua come sopra, verificando quanto necessario.
   Per implementare il metodo che accetta in ingresso il file csv, si introduca il metodo enrollStudents(String name, MultipartFile file) mappato sulla URL opportuna. I due parametri devono essere etichettati rispettivamente come @PathVariable e come @RequestParam("file") . Tale metodo dapprima controlla che il content-type della risorsa inviata sia text/csv, restituendo l'errore UnsupportedMediaType in caso contrario, dopodiché invoca l'opportuno metodo del servizio per effettuare la registrazione degli studenti contenuti nel file.
   Per verificarne il corretto funzionamento, si utilizzi il comando http -f POST <url> file@./<nomefile>.csv

All'atto della formazione di un gruppo, se esistono le precondizioni per la sua costituzione (cardinalità, assenza di duplicati, iscrizione al corso di tutti i membri, non partecipazione ad

altri gruppi nell'ambito dello stesso corso, …) viene costituito un gruppo in pectore. La variabile status, al suo interno, dice che non è ancora stato confermato da tutti i partecipanti. A questo scopo, ad ogni partecipante proposto viene mandata una email con due link, uno per confermare l'altro per annullare la formazione del gruppo. Quando tutti gli studenti visitano il proprio link di conferma il gruppo passa allo stato "attivo". Se almeno uno studente visita il link di annullamento prima che il gruppo sia diventato attivo, il gruppo viene sciolto e tutti gli studenti tornano liberi nell'ambito del corso. Se entro un dato intervallo non viene effettuata nessuna scelta, il gruppo viene sciolto comunque.

8. Nel package services si crei l'interfaccia classe NotificationService contenente un singolo metodo, void sendMessage(String address, String subject, String body)
   Seguendo le indicazioni contenute nell'articolo https://www.baeldung.com/spring-email, si implementi tale servizio nella classe NotificationServiceImpl. Si verifichi il suo corretto funzionamento.
   Poiché i messaggi devono contenere dei link distinguibili, occorre generare e memorizzare dei token casuali da associare alla singole url i cui link sarà contenuto nel messaggio inviato. A tale scopo si crei nel package entities la class Token, con il campo id, di tipo String, il campo teamId, di tipo Long ed il campo expiryDate, di tipo Timestamp. Scopo di questa classe è tenere traccia dei link inviati. Il campo id verrà usato come parte delle URL da contattare.
   Si aggiunga, nel package repositories, l'interfaccia TokenRepository che estende JpaRepository e si aggiungano due metodi:
   - List<Token> findAllByExpiryBefore(Timestamp t); //per selezionare quelli scaduti
   - List<Token>findAllByTeamId(Long teamId); //per selezionare quelli legati ad un team
   Si verifichi il suo corretto funzionamento delle modifiche introdotte.

9. Si aggiungano, all'interfaccia NotificationService, due metodi:
   - boolean confirm(String token); // per confermare la partecipazione al gruppo
   - boolean reject(String token); //per esprimere il proprio diniego a partecipare
   - void notifyTeam(TeamDTO dto, List<String> memberIds);
   Nella classe NotificationServiceImpl, li si implementino, avendo cura di verificare che il token esista e non sia scaduto. In caso affermativo il token viene rimosso dalla tablella. Dopodiché, il metodo confirm(…) dovrà verificare se per il team corrente ci sono ancora token pendenti: se questa è l'ultima conferma, dovrà modificare lo stato del team ad attivo (si introduca un metodo apposito nell'interfaccia TeamService) e ritornare true. In tutti gli altri casi tornerà false.
   Il metodo reject(…), viceversa, dovrà eliminare tutti i token ancora presenti relativi al team corrente, ed invocare il metodo evictTeam(teamId) dell'interfaccia TeamService e tornare true. In tutti gli altri tornerà false.
   Il metodo notifyTeam(…) dovrà generare tanti token casuali (UUID.randomUUID().toString()) quanti sono i partecipanti, usando come data di scadenza now()+1h e memorizzarli nel DB, tramite l'opportuno repository.
   Dopodiché confezionerà i messaggi contenti le URL /notification/confirm/<token> e /notification/reject/<token> e li invierà ai rispettivi destinatari derivando in modo algoritmico il loro indirizzo di posta (s<ID>@studenti.polito.it) (si forzi, nel metodo sendMessage(…) il proprio indirizzo di posta come reale destinatario)
   Si verifichi il corretto funzionamento del sistema.

10. Si aggiunga, nel package controllers, la classe NotificationController, etichiettata come @Controller responsabile di gestire le URL /notification/** e visualizzare una pagina html opportuna. Tale controllore provvederà a contattare gli opportuni metodi del servizo

Il server offrirà accesso ad un insieme di end-point il cui accesso sarà autorizzato tramite password.

11. Seguendo le indicazioni contenute negli articoli https://medium.com/@hantsy/protect-rest-apis-with-spring-security-and-jwt-5fbc90305cc5 o https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world, si implementi la componente di sicurezza mediate JWT.

Si utilizzi il meccanismo di configurazione dichiarativa delle regole di sicurezza per permettere un accesso condizionato alle URL /API/** ai soli utenti autenticati.
Si aggiungano, sui singoli metodi del servizio, le necessarie annotazioni di tipo @PreAuthorize che limitino l'uso delle funzionalità ai soli aventi diretto (in base al ruolo)
A tale fine si ricordi che chi ha il ruolo docente ha la possibilità di

- aggiungere un nuovo corso specificando la dimensione minima e massima dei gruppi al suo interno
- abilitare o disabilitare un corso (impedendo così ulteriori evoluzioni del suo stato)
- aggiungere singolarmente studenti ad un corso
- aggiungere l'elenco degli studenti iscritti ad un corso a partire da un file CSV

Uno studente, invece, può:

- Vedere l'elenco dei corsi a cui è iscritto
- Vedere l'elenco dei gruppi a cui è iscritto
- 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
- Vedere l'elenco dei gruppi di un dato corso
- Vedere la lista di studenti di un corso che partecipano già o che non partecipano ancora ad un gruppo

Si realizzi **lo strato REST** 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#
- Spring Boot Security + JWT ''Hello World'' Example https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world
- Protect REST APIs with Spring Security and JWT https://medium.com/@hantsy/protect-rest-apis-with-spring-security-and-jwt-5fbc90305cc5
- A command line HTTP client https://httpie.org/
- An Intro to Spring HATEOAD https://www.baeldung.com/spring-hateoas-tutorial
- Guide to Spring Email https://www.baeldung.com/spring-email