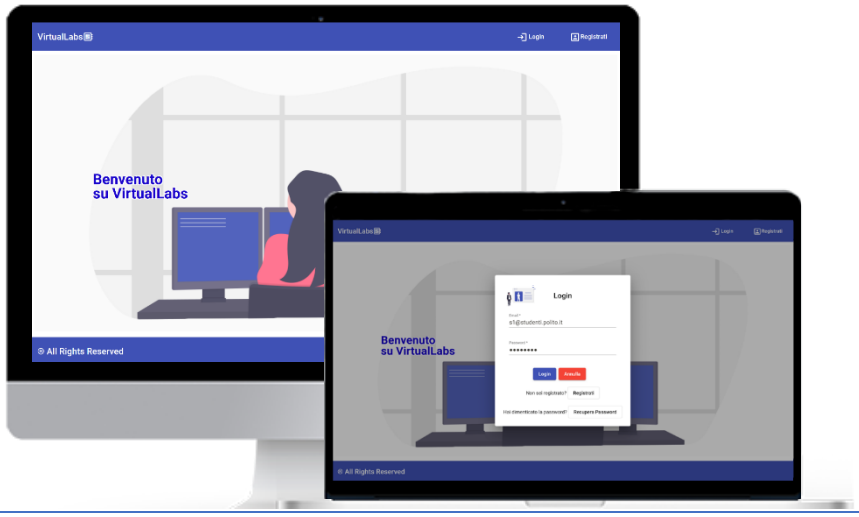

Relazione progetto VirtualLabs

ANNO ACCADEMICO: 2019/20

CLIENT SIDE: Alexandro Vassallo (266556) - Giacomo Gorga (267782)
SERVER SIDE: Valentina Margiotta (267543) - Salvatore Russo (267560)

Sommario

Introduzione del progetto

VirtualLabs è un progetto assegnato durante il corso di Applicazioni Internet del Politecnico di Torino; viene richiesta la realizzazione di una applicazione web per la gestione di laboratori virtuali in un contesto universitario.

Tali laboratori sono caratterizzati da consegne svolte da gruppi di studenti attraverso delle macchine virtuali monitorate dai docenti. Le consegne vengono assegnate da uno dei docenti referenti del corso, che avrà la possibilità di gestire le versioni degli elaborati forniti dagli studenti iscritti a tale corso.

Configurazione iniziale

Al fine di testare il funzionamento dell'applicazione sono stati inseriti i seguenti utenti:

- d1@polito.it password: 01234567
- d2@polito.it password: 01234567
- s266556@studenti.polito.it password: 01234567
- s267543@studenti.polito.it password: 01234567
- s267560@studenti.polito.it password: 01234567
- s267782@studenti.polito.it password: 01234567
- s1@studenti.polito.it password: 01234567
- s3@studenti.polito.it password: 01234567

È stato inoltre aggiunto il corso "Applicazioni Internet" avente come titolari i docenti "d1" e "d2", tutti gli studenti presenti nel sistema sono stati aggiunti al corso e sono state inoltre inserite due consegne.

È stato creato un team, denominato "Team1", avente come membri gli studenti s267543, s266556, s267560 e due VM "VM1" e "VM2".

Lo studente s267543 ha caricato una versione dell'elaborato e il docente ha caricato due revisioni.

Istruzioni per l'avvio

Il Server, così come il Client, viene avviato tramite un sistema di virtualizzazione a container attraverso l'ausilio della piattaforma open-source Docker.

A tale scopo, è stato definito un file *docker-compose.yml* in cui vengono definiti 3 servizi:

- ***Virtuallabs-server-backend***: contenitore che include l'applicazione server e si basa sull'immagine Alpine Linux. Viene definita una variabile d'ambiente denominata SPRING_DATASOURCE_URL inizializzata con l'indirizzo del servizio Virtuallabs-database. Tale variabile, viene utilizzata nel file application.properties per permettere l'accesso e l'interazione con il modello dati del nostro database. Il servizio viene esposto alla porta 8080:8080.
- ***Virtuallabs-client-frontend***: contenitore che include l'applicazione client e si basa sull'immagine node in ambiente Linux. Il servizio viene esposto alla porta 4200:4200.
- ***Virtuallabs-database***: contenitore dal quale è raggiungibile la base dati e configurato a partire da un'immagine MariaDB raggiungibile alla porta 3306:3306. Viene utilizzato un volume persistente (cartella db_data) per ottenere una configurazione iniziale del sistema.

Per avviare l'applicazione VirtualLabs, è necessario lanciare i seguenti comandi nella CLI all'interno della cartella principale VirtualLabs:

- ***docker-compose build***: si occupa di scaricare le immagini specificate nel docker-compose file
- ***docker-compose up***: per avviare l'esecuzione dei contenitori

Il client è accessibile aprendo il browser all'indirizzo ***http://localhost:4200.***

Durante l'utilizzo dell'applicazione è necessario avere accesso a degli account e-mail per completare le procedure di registrazione e conferma/rifiuto adesione ad un team. Di conseguenza, sono stati creati due account gmail, il primo utilizzato per inviare le e-mail e il secondo per riceverle. Vengono di seguito fornite le credenziali per accedere al secondo account:

email: testaivirtuallabs@gmail.com

password: testai2020!

Iniziamo ora l'esposizione del lavoro separandolo in due parti, per trattare al meglio il lato client e server dell'applicazione web.

Server side

Il server del progetto *VirtualLabs* è implementato come servizio REST basato sul framework Spring. Questo framework mette a disposizione una serie di moduli e strumenti utili che sono stati utilizzati per realizzare i diversi aspetti del servizio, tra cui Spring Boot, Spring Web, Spring Data e Spring Security. Un plugin frequentemente utilizzato è Lombok, utile per rendere più compatto il codice.

Vediamo ora come i vari moduli di Spring sono stati utilizzati, affrontando quelli che sono i componenti del progetto. Li prenderemo in esame a cominciare dal layer più basso, ovvero quello inerente alla gestione del meccanismo di persistenza, per arrivare sino al livello utente.

Gestione dei dati

La persistenza dei dati e l'accesso alla base dati relazionale vengono gestiti tramite l'interfaccia JPA che estende Repository. Per ogni entità, viene generata un'interfaccia annotata con ***@Repository*** e, nella maggior parte dei casi, non ha un'implementazione, poiché' è Spring Data a occuparsi del corretto funzionamento.

Le tabelle sono rappresentate dalle entità, ovvero delle classi Java annotate con ***@Data*** e ***@Entity***. Per ciascuna entità esiste un DTO, che contiene tutti gli attributi propri della tabella (ad eccezione degli attributi associate alle relazioni e quindi delle foreign keys), e verranno utilizzati dai controllori per tornare i dati al client.

Per quanto riguarda la gestione degli utenti, è stata definita un'entità ***User*** che tiene traccia della matricola dell'utente, nonché primary key, della password generata tramite `Bcrypt` con l'uso di un sale randomico, del ruolo dell'utente che può essere student/professor e di un flag boolean per rappresentare l'avvenuta conclusione della registrazione dell'utente al sistema.

L'entità ***TokenRegistration*** memorizza il token e la corrispondente data di scadenza che vengono utilizzati in fase di conferma dell'avvenuta registrazione. In maniera analoga, l'entità ***PasswordResetToken*** memorizza le informazioni per la gestione del recupero della password.

Per la gestione delle immagini associate alla VM, al modello VM, consegne ed elaborati è stata definita un'entità ***Image*** nella quale vengono memorizzare le informazioni delle immagini attraverso due variabili String (nameFile e type) e un vettore di byte che contiene i dati dell'immagine la quale deve avere dimensione massima di 15MB.

Le classi ***AvatarProfessor***, ***AvatarStudent***, ***PhotoModelVM***, ***PhotoVM***, ***PhotoAssignment***, ***PhotoVersionHomework*** e ***PhotoCorrection*** estendono l'entità ***Image***. Prima di essere memorizzate nel database, le immagini ricevute dal client vengono compresse e successivamente decompresse quando devono essere recuperate dalla base dati.

Le informazioni del modello VM vengono memorizzare direttamente all'interno dell'entità ***Course***, ad eccezione dell'immagine associata al modello che viene collegata con l'entità ***PhotoModelVM*** tramite una relazione 1-1.

Un ragionamento analogo è stato seguito per le restanti entità che prevedono l'utilizzo di un'immagine.

L'entità ***Team*** contiene il nome del team, la matricola del creatore e uno stato, ovvero una stringa che può assumere tre valori:

- ***Active***: se tutti gli studenti hanno accettato l'invito al team e il team è stato effettivamente creato
- ***Pending***: se uno o più studenti non hanno ancora risposto all'invito di partecipazione al team
- ***Disabled***: se uno o più studenti hanno rifiutato l'invito al team

Vengono inoltre memorizzate le risorse ancora disponibili per la creazione di VM da parte dei membri del team. Infine, l'attributo ***disabledTimestamp***, che contiene la data di disattivazione del team, permette l'eliminazione del team e dei relativi token una volta trascorsi 5 minuti dalla sua modifica.

Alla fase di creazione del team è anche collegata l'entità ***Token***, la quale tiene traccia dello stato di adesione degli studenti ai team. Tale stato, come per l'entità ***Team***, può assumere tre valori:

- ***Accepted***: se lo studente ha accettato l'invito al team
- ***Pending***: se lo studente non ha ancora dato una risposta all'invito al team
- ***Rejected***: se lo studente ha rifiutato l'invito al team

Viene inoltre memorizzato il timestamp relativo alla scadenza della proposta del team. Una volta scaduto il timestamp, lo stato del team passa a ***disabled***.

Servizi

I servizi costituiscono il modo con cui ci si interfaccia ai repository. Ogni servizio è costituito, come richiesto dal principio della ***Dependency Injection***, da un'interfaccia e da una sua implementazione. Le implementazioni dei servizi sono annotate con ***@Service*** e ***@Transactional***. I repository vengono iniettati tramite l'annotazione ***@Autowired***.

I metodi sono stati distribuiti nei servizi ***VLServiceStudent*** e ***VLServiceProfessor*** seguendo come criterio di suddivisione il ruolo dell'utente. Inoltre, nel ***VLService*** sono presenti i metodi che non richiedono un ruolo specifico, così come i metodi per la gestione delle notifiche inviate via mail per registrazione e partecipazione a un team.

Nel ***VLService***, è stato definito il metodo ***run***() il quale ogni 20 secondi si occupa di:

- Disabilitare i team dei quali è stata superata la data di scadenza della proposta
- Eliminare i team disabilitati da più di 5 minuti e i rispettivi token
- Rendere non più modificabili le consegne scadute
- Eliminare gli utenti che non hanno attivato il loro account tramite e-mail entro 5 minuti dalla loro registrazione sul sistema

Troviamo poi anche ***AuthenticationService*** e ***UserDetailsService***: il primo per la gestione della registrazione e validazione del token, il secondo per il salvataggio di utenti e per reperire le informazioni tramite l'opportuno repository; quest'ultimo servizio implementa un'interfaccia fornita da Spring Security.

Controllori

I controllori sono le classi che si occupano di intercettare le richieste entranti, mettendo a disposizione gli endpoint HTTP grazie ai quali un client può effettuare le richieste, chiamare le funzioni corrette dello strato di servizio e popolare la vista con i dati derivanti. Vengono annotati con ***@RestController*** e ***@RequestMapping***(…): quest'ultima definisce la "radice" delle API comune a tutti i metodi della classe. Ciascun metodo avrà un'annotazione ***@GetMapping***(…), ***@PostMapping***(…) o ***@DeleteMapping***(…) a seconda del tipo di richiesta HTTP che deve servire (rispettivamente GET, POST e DELETE) e che specifica il restante path dell'API.

I servizi vengono qui iniettati con ***@Autowired***.

I controllori presenti sono:

- ***UserController***: gestisce le operazioni di creazione di un nuovo utente, di cambio password e aggiornamento avatar utente. Per quanto riguarda il metodo '***registerUser'*** verrà discusso nella sezione inerente alla sicurezza. Considerando i metodi '***confirmationPage'*** e '***showChangePasswordPage'***, entrambi sono chiamati dal click di un link inviato via mail, e il loro valore di ritorno di tipo ***ResponseEntity*** serve a causare il redirect sul client a una specifica route, per mostrare la pagina coerente all'azione che si deve svolgere o al messaggio che si vuole mostrare. I metodi ***resetPassword***, ***showChangePasswordPage*** e ***savePassword*** sono utilizzati per il recupero della password mentre il metodo ***changeUserPassword*** viene utilizzato per il cambio password da parte di un utente già autenticato;
- ***ProfessorController***: mappa tutte le richieste HTTP che iniziano con "/API/professors", gestisce solo operazioni che possono essere svolte da un utente con il ruolo di professore oltre i metodi per recuperare le informazioni associate al professore;
- ***StudentController***: mappa tutte le richieste HTTP che iniziano con "/API/students", gestisce solo operazioni che possono essere svolte da un utente con il ruolo di studente oltre i metodi per recuperare le informazioni associate allo studente;
- ***CourseController***: mappa tutte le richieste HTTP che iniziano con "/API/courses", contiene gli endpoint per i metodi che riguardano la gestione di un corso, come creazione, eliminazione, modifica, ma anche il reperimento di relative informazioni. Alcune di queste API possono essere chiamate solo da un professore, altre solo da uno studente, altre da entrambi;
- ***TeamController***: mappa tutte le richieste HTTP che iniziano con "/API/teams", contiene gli endpoint per i metodi che riguardano la creazione e la gestione di un team del corso. In particolare, la creazione di un team richiede una determinata procedura: lo studente creatore deve effettuare una proposta di partecipazione ai colleghi, i quali dovranno accettare entro una scadenza. Una proposta è costituita da nome del team, scadenza, nome del creatore e lista dei partecipanti; a ciascuno studente, sia creatore che partecipante, è associato uno stato, attributo status dell'entità ***Token*** discusso in precedenza. La creazione del team avviene automaticamente all'atto di creazione della proposta, ma la sua effettiva attivazione avviene dopo che tutti gli invitati abbiano accettato la proposta; un solo rifiuto comporta la disabilitazione automatica del team, con conseguente cancellazione dopo 5 minuti. Non è possibile modificare i partecipanti, né eliminare il team. In caso di eliminazione

di uno studente dal corso, si possono presentare più scenari: nel caso in cui il team si trovi in fase di creazione, esso viene eliminato; se il team è già attivo, viene rimosso solamente lo studente; nel caso in cui lo studente rimosso fosse l'ultimo rimasto nel team allora quest'ultimo viene automaticamente eliminato. Le API fornite da questo controllore possono essere in parte chiamate solo da uno studente, altre da entrambi i ruoli. Inoltre, tale controllore contiene gli endpoint per conferma o rifiuto partecipazione ad un team;

- ***JwtAuthenticationController***: gestisce la fase di login.

I metodi del controllore possono tornare void, un DTO, oppure una Map<String, Object> (o List<Map<String, Object>); quest'ultima è utilizzata per gestire valori di ritorno più complessi, e/o quando le informazioni necessarie al client non sono rappresentate tramite nessun'entità.

Gestione delle eccezioni

Ogni operazione richiesta non consentita, viene gestita con un'opportuna eccezione. La loro gestione è importante sia per uno sviluppatore che per un utente, perché tramite il messaggio d'errore associato a tale eccezione si può capire cosa non ha funzionato. Tutte le eccezioni utilizzate nei servizi estendono ***VLServiceException***, la quale a sua volta estende ***RuntimeException***.

I metodi nei controllori, quando catturano un'eccezione custom proveniente da un servizio, lanciano a loro volta una ***ResponseStatusException***, che riceve come parametri il tipo di errore HTTP riscontrato (`401 Unathorized`, `403 Forbidden`, ecc.), stabilito sulla base dell'eccezione catturata (ad esempio, una ***StudentNotFoundException*** sarà associata a un `404 Not Found`), e il messaggio associato all'eccezione stessa.

Questo meccanismo è essenziale anche lato client non solo per mostrare un alert message all'utente con l'errore, ma anche per effettuare una corretta trattazione dell'errore HTTP riscontrato con opportune azioni.

Sicurezza

Per fornire una panoramica completa di ciò che riguarda la sicurezza, iniziamo descrivendo dettagliatamente cosa accade quando un utente effettua registrazione e login.

Per quanto riguarda la fase di registrazione, l'utente compila il form fornito dal client inserendo nome, cognome, matricola, e-mail, password e una foto profilo. Il metodo che espone l'endpoint "***/addUser***" è ***registerUser*** presente nello ***UserController.*** Qui viene definito il ruolo del nuovo utente, che può essere "***student***" o "***professor***", e viene discriminato in base all'email. Si noti come siano ammesse soltanto le email istituzionali del Politecnico, quindi con dominio @polito.it o @studenti.polito.it e pertanto viene verificato che la prima parte dell'e-mail nel primo caso sia una matricola che identifica un docente (inizia con 'd'), nel secondo che identifica uno studente (inizia con 's'). Il ruolo è importante per limitare all'utente solo alcune funzionalità. Quindi, viene creato lo user, lo studente/professore e trattato l'avatar, e il tutto salvato nei rispettivi repository. All'atto della creazione, viene generato un token corrispondente all'utente, che verrà incluso nel link inviato via mail da cliccare entro 5 minuti per confermare l'avvenuta registrazione.

Un ulteriore aspetto che si vuole sottolineare riguarda la password: il server ne riceve l'MD5 (N.B.: è stato utilizzato questo algoritmo di cifratura per semplicità, ma deve essere inteso genericamente come un metodo per non inviare la password in chiaro), e nel DB viene salvata un'ulteriore cifratura dell'MD5, per garantire un livello di sicurezza maggiore.

Per quanto concerne il login, l'utente, lato client, compila un form inserendo e-mail e password e clicca sul pulsante di login per inoltrare la richiesta al server. Il metodo ***createAuthenticationToken*** del ***`JwtAuthenticationController`***, che espone l'endpoint "***/login***", ha come parametro un oggetto di tipo ***JwtRequest***, che ha 2 campi username e password (sui quali viene mappato il JSON contenuto nella richiesta HTTP), ed effettua per prima cosa un controllo di validità sull'email ricevuta in input. Successivamente, entrano in gioco ***l'AuthenticationManager*** e altre interfacce fornite da Spring Security, che si occupano della verifica delle credenziali, ovvero che la password effettivamente coincida con quella dell'utente con id specificato. Se la verifica va a buon fine, tramite lo ***`UserDetailsService`*** si ottengono tutte le informazioni relative all'utente di cui si è inserito le credenziali. Infine, si deve generare il token di tipo JWT: esso è di fatto costituito da una stringa codificata con HMAC basata su SHA-512, che contiene i ***claims***, ovvero una mappa con ruolo, nome, cognome e id dell'utente, e un campo che ne definisce il ***timeout*** (settato a 1 ora dal momento della generazione). Se durante tutta questa procedura non si solleva alcuna eccezione, il client riceve una risposta HTTP 200 OK con il token appena generato nel body, che verrà salvato e utilizzato per autenticare tutte le successive richieste.

Client side

Il client di VirtualLabs è una Single Web Application realizzata con il framework Angular.

La procedura qua descritta rappresenta solo una traccia da seguire per verificare il funzionamento base e per puntualizzare alcuni dettagli implementativi che si basano su assunzioni fatte dal gruppo di lavoro per gestire determinati aspetti richiesti dalla specifica.

La home dell'applicazione presenta una schermata di benvenuto, in cui le uniche operazioni che possiamo svolgere sono quelle di registrazione e di login. La fase di registrazione richiede i dati di un ipotetico studente/professore del Politecnico di Torino, in quanto è vincolante che la mail inserita appartenga al dominio @polito.it oppure @studenti.polito.it.

La conferma della registrazione avviene cliccando un link ricevuto via mail, che rimanderà a una pagina che ci inviterà ad effettuare il login con l'utente appena creato. Un utente può avere il ruolo di **student** o di **teacher**, e chiaramente in base a tale ruolo avrà privilegi e funzionalità a disposizione differenti.

Per poter effettuare delle operazioni, occorre innanzitutto creare alcuni utenti con diversi ruoli. A valle di ciò, accediamo con un utente teacher, e creiamo un corso premendo sul pulsante "Crea corso" presente nella sidebar di sinistra e compilando i campi con le informazioni richieste, ed eventualmente aggiungendo altri insegnanti (che avranno gli stessi privilegi del creatore). A seguito della creazione, saremo subito reindirizzati alla pagina per l'aggiunta degli **studenti**, che può avvenire facendo una ricerca globale nella barra di ricerca, o tramite un file CSV.

Accediamo ora con un utente student che abbiamo precedentemente aggiunto al corso: aprendo la sidebar sulla sinistra, troveremo una entry con il nome del corso, e cliccandoci sopra saremo reindirizzati automaticamente alla pagina dove potremo gestire i **gruppi**. Possiamo creare una proposta di partecipazione cliccando sul pulsante "Nuovo gruppo": ci viene richiesto come vogliamo chiamare il gruppo, la scadenza della richiesta (di default a 1 settimana), e di aggiungere studenti tra quelli iscritti al corso, stando nel range scelto dal professore.

Ciascun membro riceverà una notifica via e-mail informandolo della richiesta di partecipazione; recandosi nella loro sezione dedicata al gruppo, troveranno i dettagli della richiesta con il nome del gruppo, il creatore, gli altri partecipanti, e i pulsanti per accettare o rifiutare.

Il gruppo sarà effettivamente creato quando tutti i membri invitati avranno accettato la richiesta; se anche solo un membro rifiuta la proposta, gli altri invitati non potranno più interagirci, ma vedranno una scritta che li informa che la proposta è disabilitata.

Appartenere ad un gruppo è importante per poter creare ed utilizzare le **macchine virtuali**, che serviranno a loro volta per poter svolgere un elaborato. Ad una macchina virtuale sono assegnate un certo numero di risorse per il suo utilizzo: la somma delle risorse di tutte le VM di un team non deve superare i limiti imposti dal professore per il corso a cui appartengono.

Con il professore possiamo creare delle **consegne**, caratterizzate da un nome, la scadenza, e un testo (che di fatto in questo progetto corrisponde a un'immagine, esattamente come le VM). La creazione di una consegna comporta la generazione automatica di **homeworks** (elaborati)**,** uno per ciascuno studente iscritto al corso.

Ciascuno studente troverà quindi nella propria sezione “Elaborati” una tabella con le informazioni su ciascuna consegna, e quindi la possibilità di caricare delle versioni del suo elaborato. Il professore può caricare una correzione se esiste almeno una versione per quell’elaborato, ed eventualmente assegnare un voto (che rende quindi l’elaborato concluso); se esistono più versioni, all’atto del caricamento la correzione viene automaticamente associata alla versione più recente.

Non ci dilunghiamo sulle altre funzionalità, delle quali avremo modo di parlare successivamente.

Struttura generale

Nella cartella src/app del client, troviamo una serie di sotto direttori che permettono la navigazione tra i componenti più rapida, e li dividono per categoria. A loro volta, possiamo trovare ulteriori sottocartelle che racchiudono aspetti ancora più specifici di quella categoria.

Di seguito analizzeremo principalmente le cartelle **student** e **teacher**, che costituiscono il raggruppamento dei componenti rispettivamente per i ruoli studente e insegnante.

Prima però di dare una descrizione dettagliata del contenuto delle cartelle, esaminiamo come è organizzato il routing, per comprendere meglio come i vari componenti vengono istanziati e incastrati tra di loro.

Routing

Nel file ***app-routing.module***, troviamo la definizione delle varie routes che compongono l’applicazione, che rendono, grazie alla funzionalità del routing di Angular, la navigazione più fluida e piacevole.

Descriviamo ora nel dettaglio ciascuna route:

- ***‘home’:*** la route di partenza, istanzia la pagina inziale tramite il componente ***HomeComponent***;
- ***‘register/confirmation ’***: corrisponde alla pagina di avvenuta registrazione, costruita con il ***RegisterSuccessComponent.*** In questa route sarà sempre presente un query param che indicherà il token di registrazione, rispettivamente “confirmToken” se il token è valido, “expToken” viceversa;
- ***‘user/password-reset’***: quando un utente dimentica la password, può resettarla. Il server manderà una mail con un link alla URL “***user/password-reset/?tokenId=[tokenId]***”; si istanzia il ***ChangePasswordComponent***, che conterrà un form per inserire, confermare e salvare la nuova password;
- ***‘student/course/:courses’***: questa è la route madre per la navigazione tra i componenti di student. Il parametro “courses” indica il nome del corso in cui stiamo navigando, ed è parsato in modo che sia tutto minuscolo ed eventuali spazi sostituti con un “-“ (allo stesso modo viene salvato sul DB).
  Troviamo quindi alcune route figlie:

a) **“teams”,** per la visualizzazione e gestione del gruppo tramite il ***TeamsComponent***, e una route figlia “request” per la creazione di un gruppo tramite il ***RequestTeamComponent***;
b) “***vms***”, per la visualizzazione e gestione delle macchine virtuali tramite il ***VmsComponent***;
c) “***assignments***”, per la visualizzazione delle consegne tramite il componente ***AssignmentsComponent***, e ha una sotto route per la visualizzazione dei dettagli dell’elaborato e caricamento versioni tramite il ***VersionsComponent***;

- ***‘teacher/course/:courses’***: analoga alla precedente, ma cambiano le route figlie:
  a) “***students***”, per visualizzazione e gestione degli studenti iscritti al corso tramite lo ***StudentsComponent***;
  b) “***vms***”, per la visualizzazione dei team del corso tramite il ***VmsComponent***, e con una sotto route “team/:idT”, dove il parametro “idT” indica l’id del gruppo selezionato per la visualizzazione delle sue macchine virtuali tramite il ***TeamVmsComponent***;
  c) “***assignments***”, per la visualizzazione e creazione delle consegne tramite il componente ***AssignmentsComponent***, e ha una sotto route del tipo “:***idA/homeworks***”, dove “idA” corrisponde all’id della consegna selezionata, ed è mostrata la tabella con l’elenco degli elaborati degli studenti con le relative informazioni tramite il componente ***HomeworksComponent***, e un’ulteriore nested route del tipo “:idH/versions”, dove “idH” corrisponde all’id dell’elaborato selezionato e vi è una vista per consultazione di versioni/correzioni e caricamento di correzioni tramite il ***VersionsComponent***.

Componenti

Abbiamo appena menzionato parlando del routing i principali componenti dell’applicazione.

Il primo componente istanziato al caricamento dell’applicazione è ***l’AppComponent***, che gestisce la mat-toolbar in alto con i pulsanti per registrazione/login/logout/profilo, la mat-toolbar con il nome del corso corrente, il routing verso i principali componenti, e l’apertura delle principali dialog.

Le ***dialog*** costituiscono una parte importante dell’applicazione, in quanto permettono la creazione e la modifica di oggetti ed entità che vengono poi visualizzati dai componenti “a schermo intero”. L’apertura di una dialog è causata di fatto da un redirect, in quanto viene aggiunto nella route un queryParam per tenere traccia dello stato (ad esempio, l’apertura della dialog di login aggiunge alla route corrente ?doLogin=true, così un eventuale refresh della pagina causa la riapertura automatica della dialog). Per ciascuna dialog, vi è un componente apposito, che elenchiamo qua di seguito:

- ***LoginDialogComponent***: contiene un form per il login con 2 input per email e password, e 2 pulsanti per reset della password o per aprire la dialog di registrazione;
- ***RegisterDialogComponent***: contiene un form per richiedere nome, cognome, matricola, e-mail, password e una foto profilo, e un pulsante per aprire la dialog di login;
- ***EditProfileComponent***: si possono visualizzare le proprie informazioni, modificare la foto profilo e cambiare la password;
- ***ForgotPasswordComponent***: contiene un input per inserire la matricola dell’utente di cui si vuole resettare la password;

- ***ViewImageComponent***: questo è un componente "versatile", nel senso che serve per visualizzare un'immagine generica. Esso richiede in fase di istanziazione alcuni parametri, tra cui una stringa "mode", che può assumere il valore "assignment", "version", "correction" e "vm". In base al mode, viene caricata l'immagine opportuna utilizzando anche gli altri parametri;
- ***AddCourseDialogComponent*** (solo teacher): serve per creare un corso nuovo, contiene un form per inserimento di nome, acronimo, massimo e minimo partecipanti gruppo, modello e foto iniziale VM*, e aggiunta professori;
- ***EditCourseComponent*** (solo teacher): vi sono i pulsanti per abilitare/disabilitare ed eliminare il corso corrente, un form per la modifica massimo e minimo partecipanti gruppo, e una tabella per l'aggiunta di altri professori;
- ***CreateAssignmentComponent*** (solo teacher): serve per creare una consegna, contiene un form che richiede nome della consegna, scadenza, e una foto che rappresenta il testo della consegna;
- ***UploadCorrectionComponent*** (solo teacher): serve per caricare la correzione inerente a una versione dell'elaborato, contiene un input che si può riempire con un numero compreso tra 0 e 30 se si ritiene essere la versione definitiva, e un input per caricare una foto che rappresenta il testo della correzione;
- ***ManageModelComponent*** (solo teacher): contiene un form con i campi del modello della VM già valorizzati, ma ovviamente modificabili e salvabili;
- ***AddHomeworkComponent*** (solo student): serve di fatto per caricare una immagine che rappresenta la versione dell'elaborato inerente a una consegna selezionata;
- ***CreateVmsComponent*** (solo student): serve per creare una macchina virtuale, contiene un form che richiede il nome della vm, e le risorse da allocare (ram, disco, vcpu);
- ***ManageVmComponent*** (solo student): può essere aperta solo dagli owners della macchina virtuale, serve per modificare i parametri configurati durante la creazione della VM, e ad aggiungere eventuali altri owners.

La quasi totalità dei componenti elencati hanno sono di fatto "sdoppiati", in quanto vi è il componente smart e il componente dummy: il primo contiene l'interazione con i servizi nonché la valorizzazione dei dati da visualizzare nel template, il quale è gestito dal componente dummy.

(*): Il modello VM viene definito all'atto della creazione del corso con valori di default, che è possibile modificare successivamente. Tuttavia, viene richiesto di inserire manualmente un'immagine iniziale.

Servizi

In Angular, i servizi sono importanti in quanto fungono da "ponte" tra il client e il server, che nel nostro caso è costituito da un servizio REST.

Nell'applicazione, vi sono 5 servizi:

- ***AuthService***: questo servizio si distingue dagli altri in quanto non fa rifermento a un controllore particolare del server, bensì raggruppa una serie di metodi (anche non riferiti ad alcun endpoint REST) che sono utili per gestire opportunamente i meccanismi di sessione e autenticazione, e un EventEmitter utile per notificare ai vari comportamenti l'avvenuto login/logout di un utente;
- ***CourseService***: fa riferimento al CourseController del server, contiene tutti i metodi corrispondenti a ciascun endpoint, un BehaviorSubject utile per tenere traccia del corso in cui si sta navigando, e due EventEmitter per notificare operazioni da svolgere sull'elenco dei corsi;
- ***StudentService***: fa riferimento allo StudentController del server, contiene tutti i metodi corrispondenti, un oggetto di tipo Student per tenere traccia dello studente loggato (vale null se l'utente loggato è un professore), due EventEmitter per notificare operazioni da svolgere sulle macchine virtuali, e un EventEmitter per notificare il caricamento di una versione elaborato;
- ***TeacherService***: fa riferimento al ProfessorController del server, contiene tutti i metodi corrispondenti, un oggetto di tipo Teacher per tenere traccia del professore loggato (vale null se l'utente loggato è uno studente), un EventEmitter per notificare la creazione di una consegna, e un altro EventEmitter per notificare il caricamento di una correzione relativa a una versione elaborato;
- ***TeamService***: fa riferimento al TeamController del server, contiene tutti i metodi corrispondenti, e un EventEmitter che serve per notificare la creazione di una proposta di partecipazione a un gruppo.

Per fare riferimento agli endpoint, tutti i servizi includono il modulo HttpClient.

I servizi, nonostante non contengano soltanto metodi, sono da considerarsi *stateless*, in quanto non viene salvata alcuna informazione se non quelle indispensabili per la gestione delle viste e della sessione.

In questa sezione menzioniamo anche l'AuthInterceptor, che arricchisce le richieste http in uscita con il token di autenticazione (se presente), e l'AuthGuard, che verifica la validità temporale del token (ogni token scade dopo 1 ora).

Templates

I template HTML, così come i componenti dummy, sono implementati sulla base della documentazione fornita sul sito https://material.angular.io , utilizzando le API e i moduli del material design.

Le viste sono organizzate in ***mat-card***, rendendole ordinate e di facile utilizzo; gli input testuali sono decorati con mat-input, i pulsanti con mat-button (piuttosto che mat-raised-button o mat-stroked-button), le tabelle con mat-table, e così via.

Quasi tutte le tabelle possiedono un meccanismo di ordinamento per ciascuna colonna, e le più grandi possiedono anche la paginazione delle righe.

Gestione degli errori

Le operazioni non consentite e i messaggi di errori ricevuti dal server vengono notificati all'utente principalmente tramite alert del browser. I messaggi sono contenuti nel campo *error* della risposta http. Gli errori vengono lanciati a seguito di un'eccezione lato server, che in base al tipo ne determina il codice e il messaggio.

In base al codice errore, viene eseguita una trattazione dedicata. In particolare, per gli errori http 404 e 403 viene effettuato un redirect alla home e una gestione a seconda della situazione.

Il 401 viene evitato grazie alla presenza dell'AuthGuard.

Scelte implementative

Per confermare l'avvenuta registrazione al sistema, l'utente è tenuto a concludere tale procedura aprendo il link presente nell'e-mail di notifica entro 5 minuti dalla compilazione del form di registrazione.

Il modello VM associato al corso viene definito dal docente direttamente in fase di creazione del corso stesso tramite apposito form. È tuttavia possibile modificarne i parametri successivamente nella vista "VMs" tramite apposito bottone.

Per quanto riguarda l'eliminazione di uno studente dal corso, se lo studente eliminato è owner di una VM, automaticamente i restanti membri del team ne acquisiscono il titolo. Inoltre, vengono automaticamente rimossi gli elaborati, le versioni e le correzioni legate allo studente stesso.

Nel caso in cui uno studente venga aggiunto ad un corso in cui sono già presenti delle consegne, viene automaticamente associato allo studente un elaborato per ogni consegna con stato a NULL.

È possibile aggiungere gli studenti ad un corso tramite file CSV che contiene i campi della classe ***StudentDTO*** ovvero id, firstName, name e e-mail. In particolare, tali valori devono coincidere con quelli associati a studenti già presenti nel sistema e quindi memorizzati nello ***studentRepository***, altrimenti viene lanciata l'eccezione ***InfoStudentsCSVWrongException***.

Quando scade una consegna, per tutti gli studenti che non hanno caricato ancora una versione dell'elaborato viene automaticamente assegnato un voto pari a 0. Lo stesso vale per gli elaborati associati a consegne già scadute nel momento in cui uno studente viene aggiunto al corso. Inoltre, lo studente a consegna scaduta può comunque visualizzare il testo della stessa ma lo stato rimane invariato (anche in caso di stato NULL).

È possibile la creazione di team composti da una sola persona nel caso in cui il docente lo permetta. In questo caso, il team viene automaticamente attivato quando uno studente compila la proposta di creazione team senza inserire altri studenti.

Il docente può assegnare un voto alla versione solo se nella sua vista è presente l'ultima versione dell'elaborato caricato dallo studente.

Conclusioni

Abbiamo affrontato come viene realizzata una versione semplificata di un'applicazione web, in questo caso per la gestione di corsi universitari e macchine virtuali. Oggigiorno, Spring e Angular sono, rispettivamente per server e client, 2 tra i framework più utilizzati anche in ambito professionale, ragion per cui è stato importante l'apprendimento e soprattutto la messa in pratica di queste metodologie di sviluppo in ambito distribuito.

Si ringrazia il lettore per l'attenzione.