Esercitazione 5

Studente: Michele Luigi Greco - s256665

Comandi da lanciare nella directory principale

- Run `npm install`
- Run `json-server-auth db.json`
- Run `npm start`

La pagina è adesso disponibile alla url: localhost:4200

Struttura dati

- L'elenco degli studenti disponibili nella collection può essere trovato all'interno del file db.json.Viene riportato un solo studente per brevità:

  "students": [{ "id": "s267612", "name": "Andrea", "firstName": "Saltarelli", "group": "Prova", "courseId": 1 }]

- All'interno dello stesso file sono riportati l'unico corso gestito attualmente (Applicazioni Internet) e un gruppo d'esempio (Calvary):

  "courses": [{ "id": 1, "name": "Applicazioni Internet", "path": "applicazioni-internet" }]

  "groups": [{ "id": 1, "name": "Calvary" }]

- Per ultimo è riportato l'utente di default il cui ID è olivier@mail.com e PASSWORD bestPassw0rd:

  "users": [{ "email": "olivier@mail.com", "password": "$2a$10$9SFIVF0tDrwRoUlZOo9Q0urEs0cnpBXu9RWGf/TxuHjFzDteQhvDW", "id": 1 }]

Gestione delle Route

All'interno del file `app-routing.module.ts` sono configurate le route e le viste del website.

```
 // Vista HomePage
{ path: 'homepage', component: HomeComponent },

// Vista StudentComponent
{
  path: 'teacher/course',
  children: [
    {
      path: 'applicazioni-internet/students',
      component: StudentsContComponent,
      canActivate: [RouteGuard]
    }
  ]
},

// Vista VMs Component
{
  path: 'teacher/course',
  children: [
    {
      path: 'applicazioni-internet/vms',
      component: VmsContComponent,
      canActivate: [RouteGuard]
    }
  ]
},
{ path: '**', component: PageNotFoundComponent }
```

- Di default il website si apre sulla HomePage, nella quale è presente un messaggio di benvenuto. La navigazione avviene come segue:
  - Cliccando sul `<button>` VirtualLabs si viene reindirizzati alla vista HomePage.
  - Cliccando sulle `<tab>` corrispondenti al nome di un corso, attualmente valido solo per il corso di applicazioni internet, si viene reindirizzati alla vista Students, dove è presente l'elenco degli studenti attualmente iscritti al corso.
  - Cliccando sulle `<tab>` Students, VMs, Assignments si verrà reindirizzati verso i rispettivi componenti, se implementati, altrimenti verso una generica pagina di errore.

Login

Come richiesto è stato implementato un auth service, reperibile nel file `auth.service.ts` che gestisce le API per il login, il server API utilizzato è json-server-auth. Per la gestione del componente login si vedano i file: - `login-dialog.component.ts` : qui viene gestita l'autentiazione, vengono reperite le credenziali del form e

chiamato l'auth manager.

```
  login(form: NgForm) {
    this.auth.login(form.value.email, form.value.password);
  }
```

- `login-dialog.component.html` : qui viene gestita invece tutta la componente grafica, ossia la `<mat-card>` dove è presente il form per effettuare il login. Sono stati effettuati dei controlli preliminari sulle credenziali: l'email deve essere valida e la password deve essere più lunga di sei caratteri, altrimenti il `<button>` del login non viene abilitato.
- `app.component.ts` : all'interno del modulo principale dell'applicazione viene invece gestita l'informazione dell'utente loggato, ossia se esso risulta essere loggato viene stampato un messaggio di benvenuto personalizzato di fianco al `<button>` del login, viceversa se esso sta effettuando il logout viene pulita la variabile locale `mail` e nascosto il messaggio.

  ```
  this.auth.user.subscribe(result => {
    if (result != null) {
      // login
      this.isUserLoggedIn = result.isLogged;
      this.email = result.email
        ? 'Welcome back ' + result.email.split('@')[0] + ' !'
        : result.email;
    } else {
      // logout
      this.isUserLoggedIn = false;
      this.email = '';
      router.navigate(['homepage']);
    }
  });
  ```

  }

Funzionalità

È stato innanzitutto implementato lo student-service, con il compito di fare da tramite per la gestione del database, che nel nostro caso non è ancora un DB vero e proprio ma un DB fantoccio, implementato tramite un file json e l'ausilio del tool json-server. Al fine di garantire compatibilità con l'API server che sarà implementato per il progetto si è forzato il ritorno di oggetti del tipo `Observable<Student[]>` e la chiamati di funzioni HTTP REST. Riportiamo le tre funzioni principali della classe:

```
query(): Observable<Student[]> {
    return this.http
      .get<Student[]>(URL_API_STUDENTS, { headers: this.getAuthHeader() })
      .pipe(map(students => students || []));
  }

updateEnrolled(students: Student[], courseId: number) {
    return from(students).pipe(
      concatMap(student => {
        student.courseId = courseId;
        return this.http.put<Student>(
          URL_API_STUDENTS + '/' + student.id,
          student,
          { headers: this.getAuthHeader() }
        );
      }),
      toArray()
    );
  }

listEnrolledStudents(courseId: number): Observable<Student[]> {
    return this.http
      .get<Student[]>(URL_API_STUDENTS + '?courseId=' + courseId, {
        headers: this.getAuthHeader()
      })
      .pipe(map(students => students || []));
  }
```

Sono stata implementate le seguenti funzionalità con persistenza dei dati: - Aggiunta di uno studente a un corso. A livello grafico si sfrutta quanto implementato nel precedente laboratorio, con la differenza che al `<button>` "Add Student" è stato preferito un `<icon-button>` del tipo `add-circle` di colore verde, altra differenza

consiste nell'invocazione dello student service per rendere persistenti le modifiche.

```
onAddStudent(student: Student) {
  this.studentService.updateEnrolled([student], 1).subscribe(s => {
    this.studentService
      .listEnrolledStudents(1)
      .subscribe(value => (this.STUDENT_DATA = value));
    this.studentService
      .query()
      .subscribe(value => (this.STUDENT_OPTIONS = value));
  });
}
```

- Rimozione di uno o più studenti da un corso. Come per la precedente funzionalità si è sfruttato quanto implementato nel precedente laboratorio, sostituendo al `<button>` "Delete Student(s)" un `<icon-button>` del tipo `delete` di colore rosso situato vicino la checkbox master. Anche in questo caso per rendere le modifiche persistenti si è sfruttato lo student service.

  onDeleteStudent(students: Student[]) { this.studentService.updateEnrolled(students, 0).subscribe(s => { this.studentService .listEnrolledStudents(1) .subscribe(value => (this.STUDENT_DATA = value)); this.studentService .query() .subscribe(value => (this.STUDENT_OPTIONS = value)); }); }

Esempio di utilizzo

Per testare a pieno il website è sufficiente svolgere i seguenti passaggi: 1. Cliccare sul `<tab>` Applicazioni Internet.

2. Effettuare il login tramite la `<mat-card>` apparsa al centro dello schermo, utilizzando come email: `olivier@mail.com` e come password: `bestPassw0rd`.
3. Verrà quindi concesso l'accesso alla vista Students, nella quale saranno mostrati alcuni studenti registrati al corso in modo hard-coded.
4. Per inserire uno studente è sufficiente cliccare sulla text-box denominata "add student" e iniziare la digitazione, se sarà riscontrato un match con quanto parzialmente digitato l'autocomplete suggerirà l'oggetto studente da aggiungere. Cliccando sullo studente questo sarà temporaneamente salvato all'interno di una variabile locale al componente, sarà committata la modifica solo quando verrà cliccato l'apposito bottone "add".
5. Per eliminare uno è più studenti è sufficiente selezionarli tramite l'apposita checkbox posta al fianco dell'ID e poi premere il bottone "delete" situato vicino la checkbox master.
6. Per verificare la persistenza dei dati si può notare che chiudendo il browser e cancellando cookie e cronologia i dati rimangono fedeli alle ultime modifiche effettuate.
7. Si fa notare, infine, che senza aver effettuato il login non è possibile effettuare alcuna modifica, ne accedere alle route protette.