Compte Rendu

Spring MVC, Spring Data JPA et Spring Security

Students Management

Tarik FERTAH

II-BDCC2

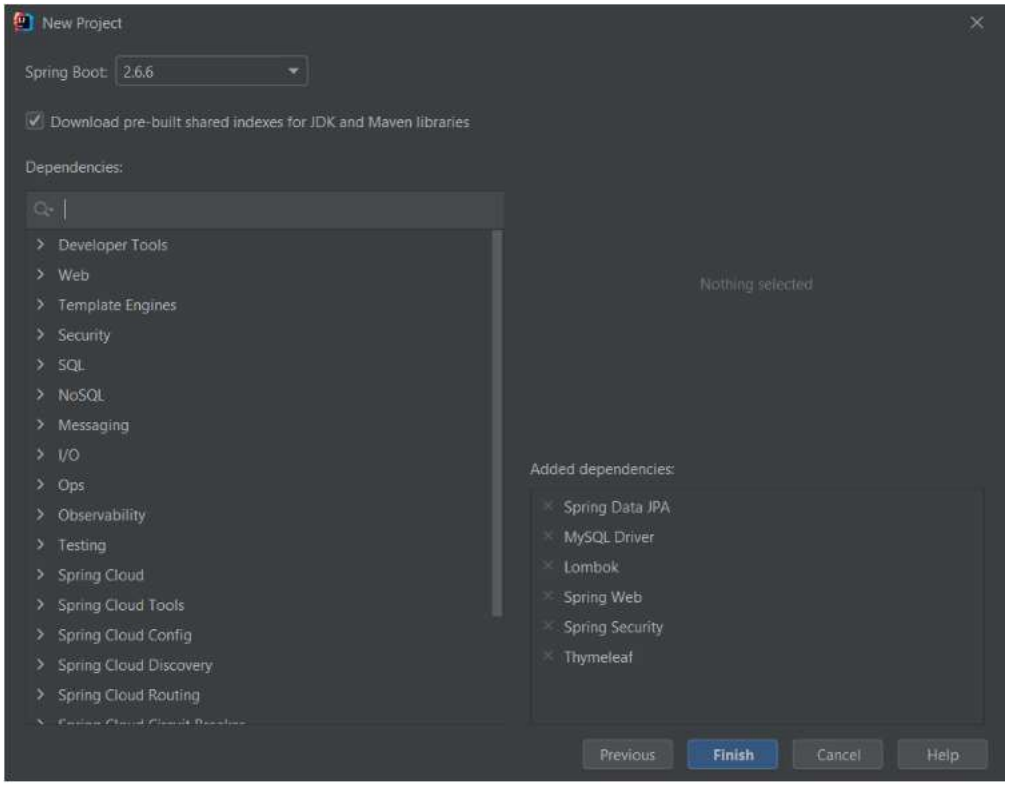
Année Universitaire : 2022 - 2023

Enancé de l’App

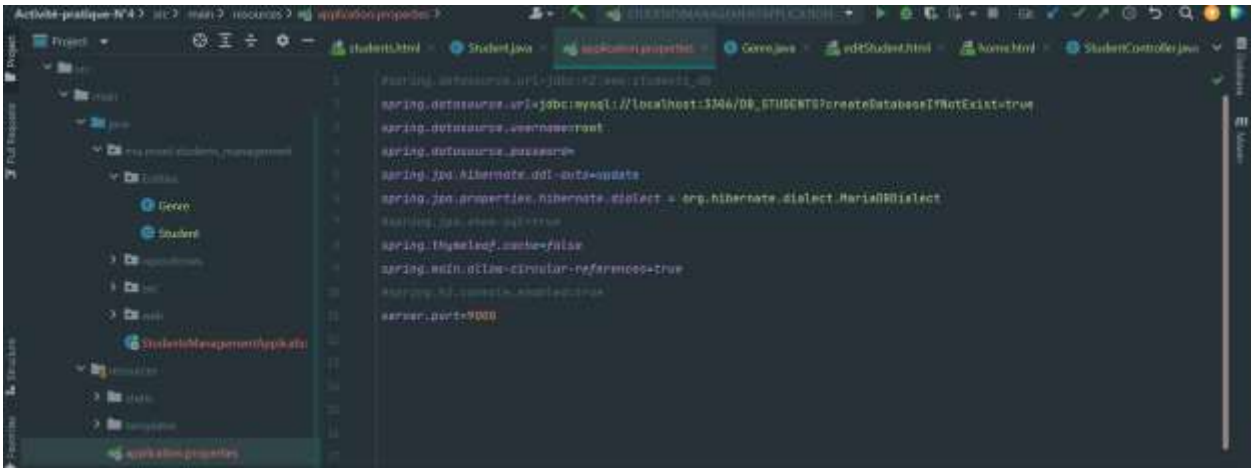
Créer une application Web basée sur Spring MVC, Spring Data JPA et Spring Security qui permet de gérer des étudiants. Chaque étudiant est défini par :

- Son id
- Son nom
- Son prénom
- Son email
- Sa date naissance
- Son genre: MASCULIN ou FEMININ
- Un attribut qui indique s’il est en règle ou non L'application doit offrir les fonctionnalités suivantes :
- Chercher des étudiants par nom
- Faire la pagination
- Supprimer des étudiants en utilisant la méthode (DELETE au lieu de GET)
- Saisir et Ajouter des étudiants avec validation des formulaires
- Editer et mettre à jour des étudiants
- Créer une page template
- Sécuriser l'accès à l'application avec un système d'authentification basé sur Spring security en utilisant la stratégie UseDetails Service

1 **Dépendances** :

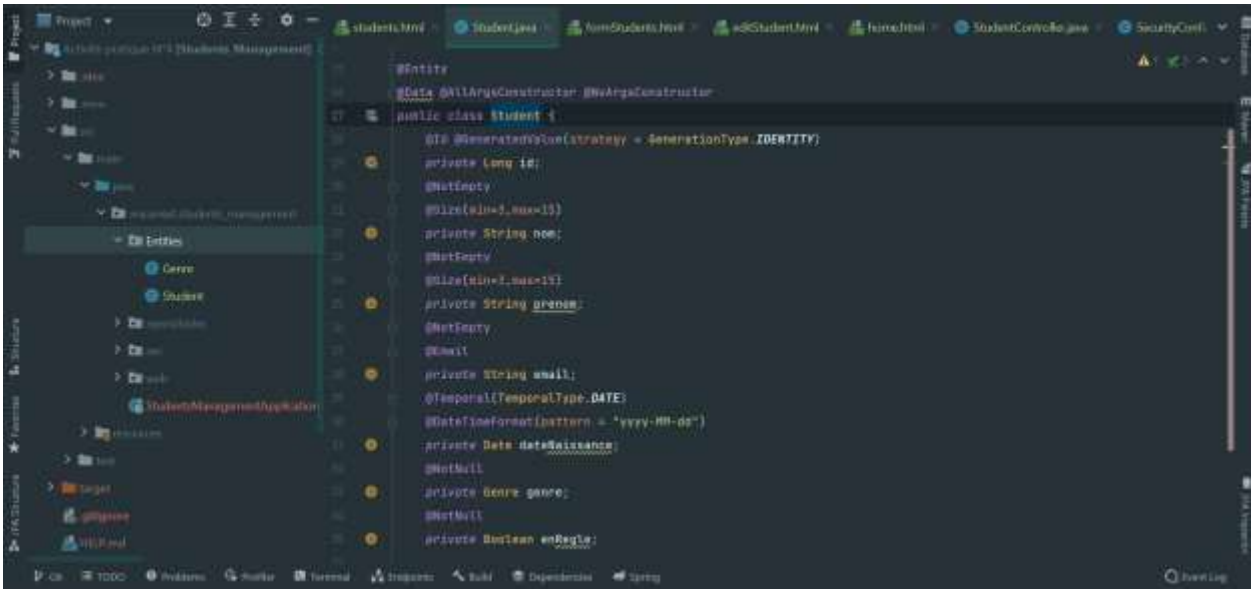


2 Configuration

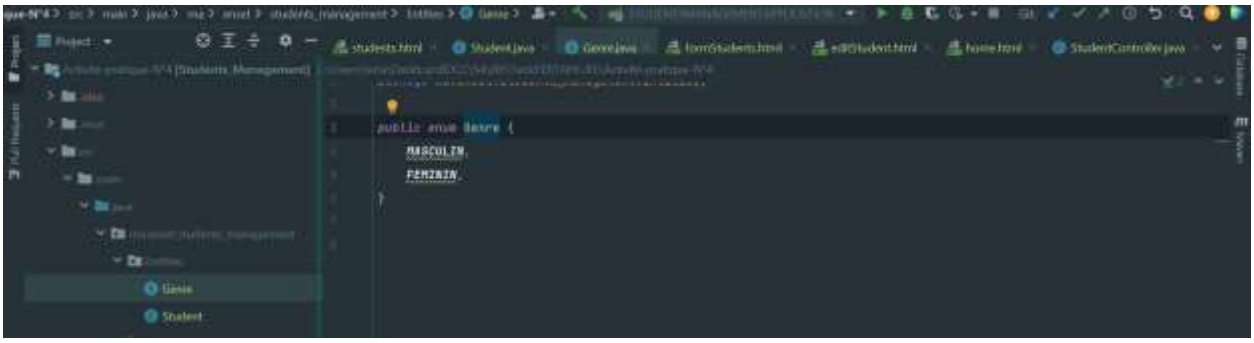


3 Création des entités

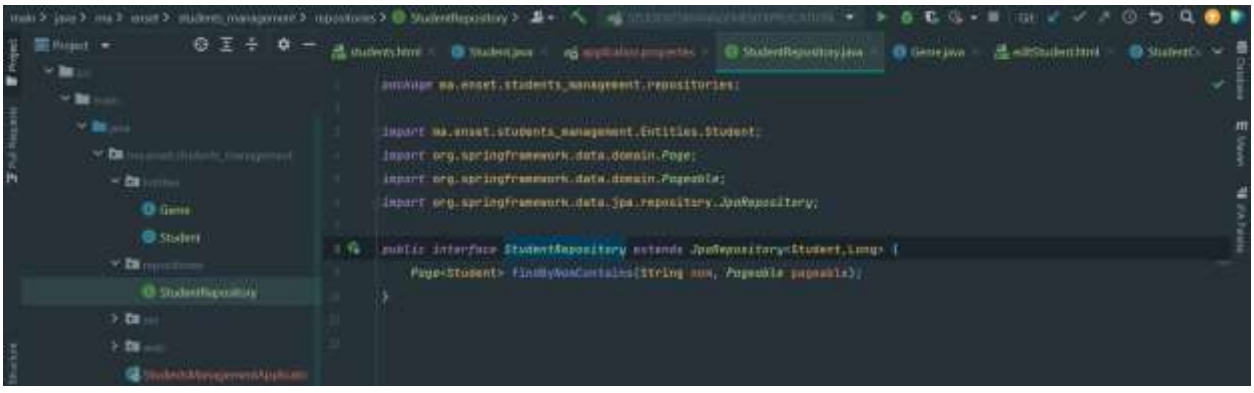
Entité « **student** » => `entities/Student`



Entité « **genre** » => `entities/Genre`



4 Création du Repository de l'entité « student » => repositories/ StudentRepository



5 Création du Controller => web/StudentController

- Home

```
@GetMapping(path = "/")
public String Home() { return "home"; }
```

- Liste des étudiant
- Recherche par keyword
- Pagination

```
@GetMapping(path = "/user/index")
public String students(Model model,
                       @RequestParam(name = "page", defaultValue = "0") int page,
                       @RequestParam(name = "size",defaultValue = "5") int size,
                       @RequestParam(name = "keyword",defaultValue = "") String keyword){
    Page<Student> pageStudents = studentRepository.findByNomContains(keyword, PageRequest.of(page,size));
    model.addAttribute( attributeName: "listStudents",pageStudents.getContent());
    model.addAttribute( attributeName: "pages",new int[pageStudents.getTotalPages()]);
    model.addAttribute( attributeName: "currentPage",page);
    model.addAttribute( attributeName: "keyword",keyword);
    return "students";
}
```

- Delete Student By Id:

```
@GetMapping(path = "/admin/delete")
public String delete(Long id, String keyword, int page){
    studentRepository.deleteById(id);
    return "redirect:/user/index?page="+page+"&keyword="+keyword;
}
```

- Add new Student

```
@GetMapping("/admin/formStudents")
public String formStudents(Model model){
    model.addAttribute( attributeName: "student",new Student());
    return "formStudents";
}
```

- Edit Student

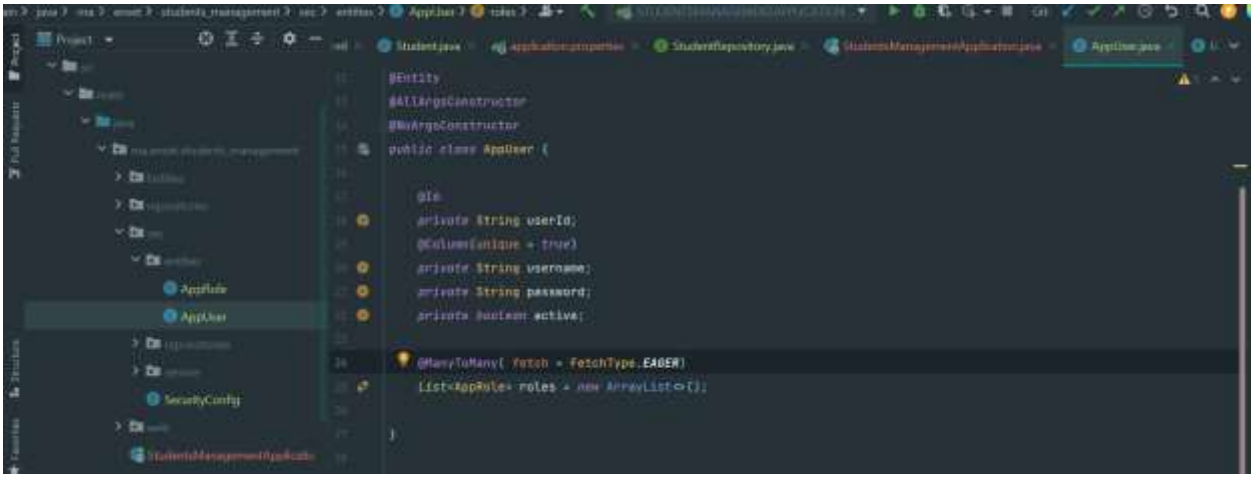
```
@GetMapping("/admin/editStudent")
public String editStudent(Model model,Long id,String keyword, int page){
    Student student = studentRepository.findById(id).orElse( other: null);
    if (student == null) throw new RuntimeException("Student introuvable");
    model.addAttribute( attributeName: "student",student);
    model.addAttribute( attributeName: "page", page);
    model.addAttribute( attributeName: "keyword", keyword);
    return "editStudent";
}
```

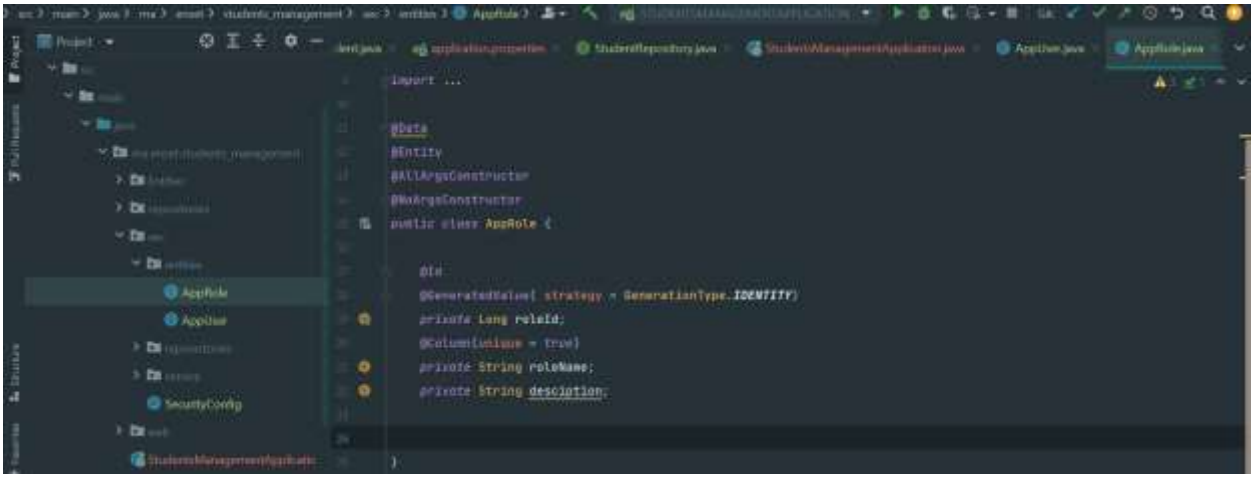
- Insertion des étudiants dans la base de données

```
public static void main(String[] args) { SpringApplication.run(StudentsManagementApplication.class, args); }
@Bean
CommandLineRunner commandLineRunner(StudentRepository studentRepository) {
    return args ->{
        studentRepository.save(new Student( id: null,  nom: "khalil",  prenom: "khalil",  email: "khalil.kha@gmail.com
        studentRepository.save(new Student( id: null,  nom: "saloua",  prenom: "saloua",  email: "saloua@gmail.com", n
        studentRepository.save(new Student( id: null,  nom: "ayoub",  prenom: "ayoub",  email: "ayoub@gmail.com", new
        studentRepository.save(new Student( id: null,  nom: "sara",  prenom: "sara",  email: "sara@gmail.com", new Dat
    };
}
```

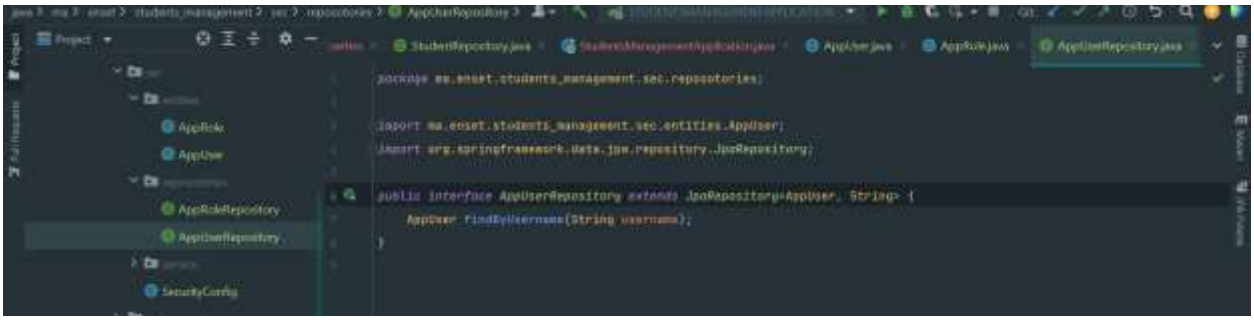
6 Spring Security

- Création des entités : Entité « User » et Entité « Role »

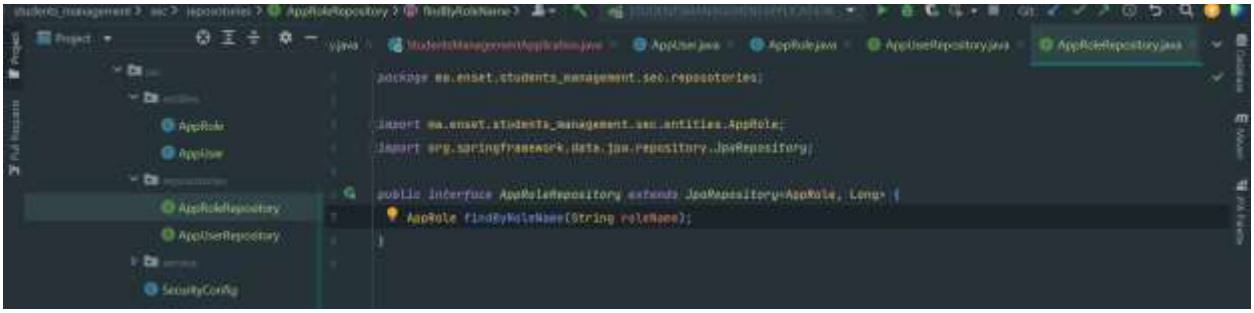




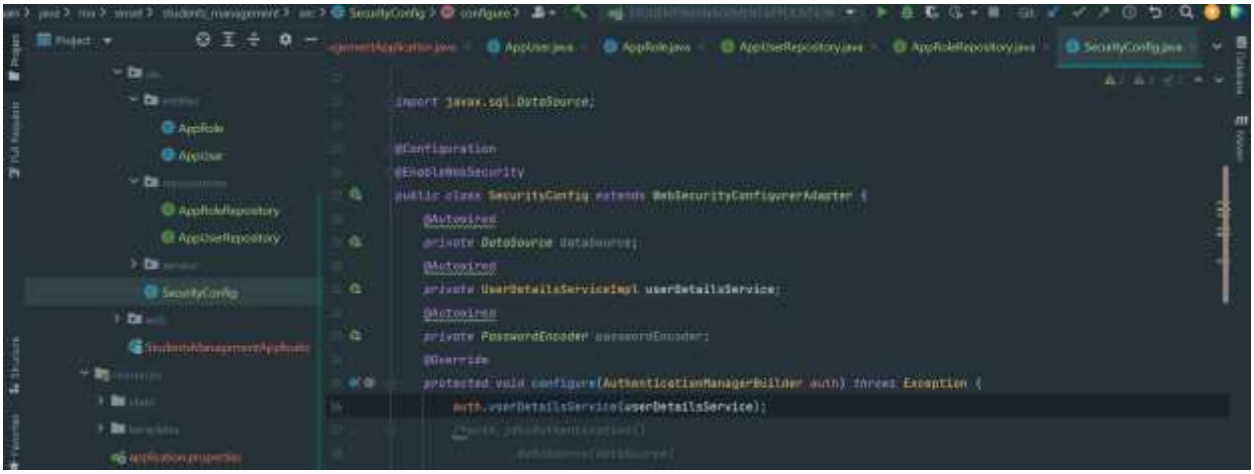
- UserRepository

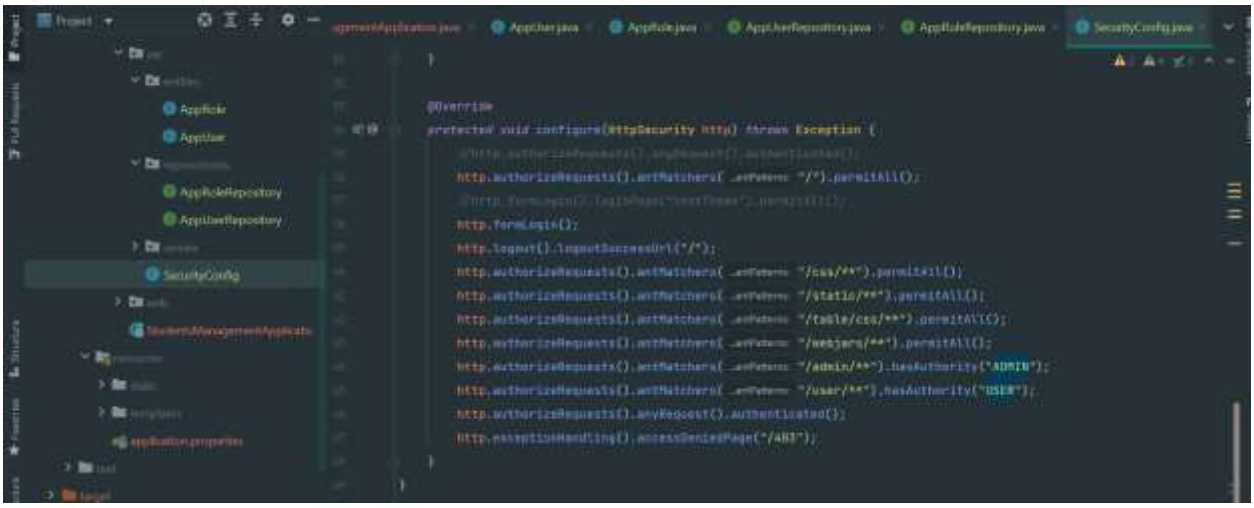


- RoleRepository



- SecurityConfig

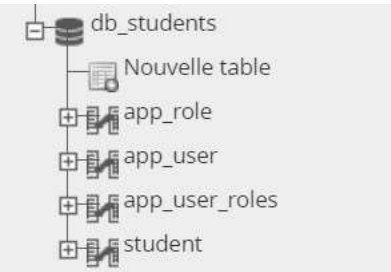




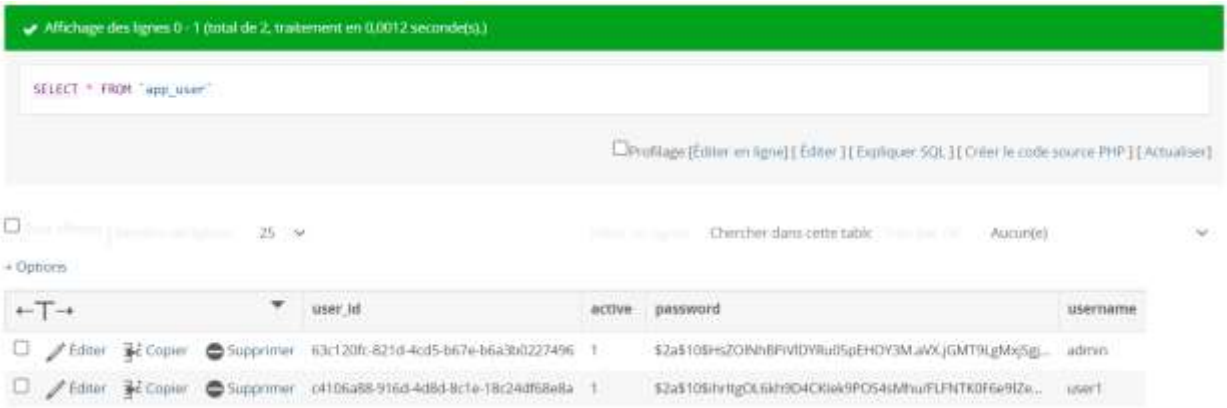
Définition de la fonction BEAN qui va fournir à l'application toujours un **PasswordEncoder** de type **`BCryptPasswordEncoder`** pour **Hasher** les mots de passe, et une deuxième fonction BEAN pour créer deux premiers utilisateurs avec deux roles **`ADMIN`** et **`USER`** :

```
@Bean
CommandLineRunner saveUsers (SecurityService securityService) {
    return args -> {
        securityService.saveNewUser( username: "admin",  password: "admin",  rePassword: "admin");
        securityService.saveNewUser( username: "user1",  password: "user",  rePassword: "user");

        securityService.saveNewRole( roleName: "ADMIN",  description: "This is the admin role !");
        securityService.saveNewRole( roleName: "USER",  description: "This is the user role !");

        securityService.addRoleToUser( userName: "admin",  roleName: "ADMIN");
        securityService.addRoleToUser( userName: "admin",  roleName: "USER");
        securityService.addRoleToUser( userName: "user1",  roleName: "USER");
    };
}
```

Au relancement de l'application, trois nouvelles tables sont crées, avec les données dedans :

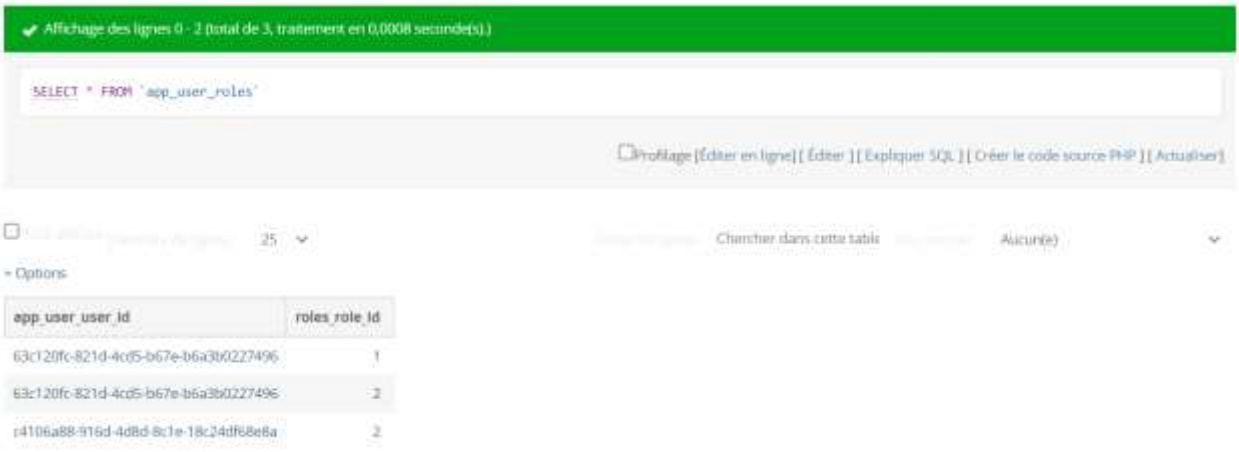


- Table « app_user »

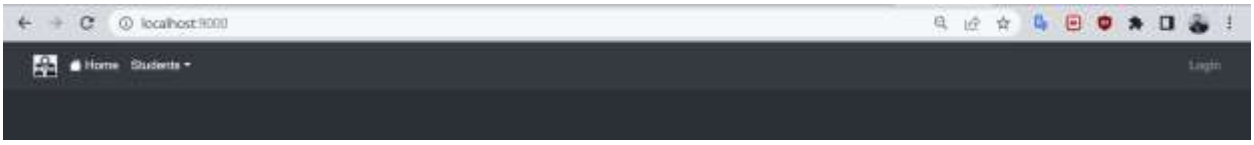


- Table « app_role »

- Table app_user_roles

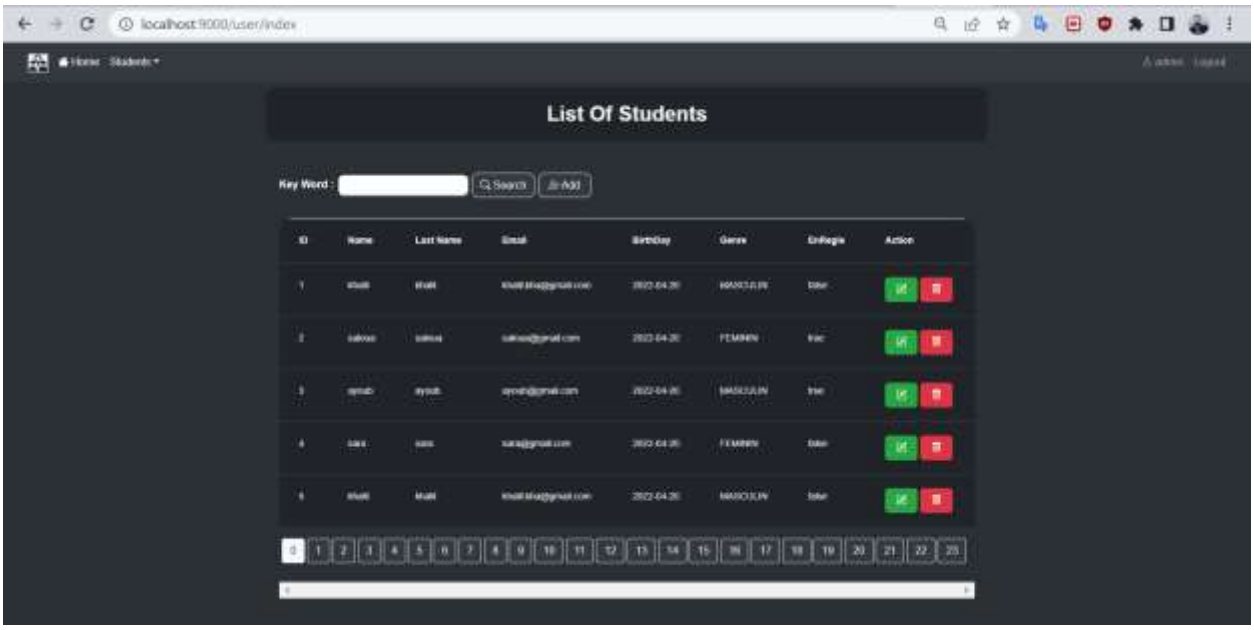


7 Page Home

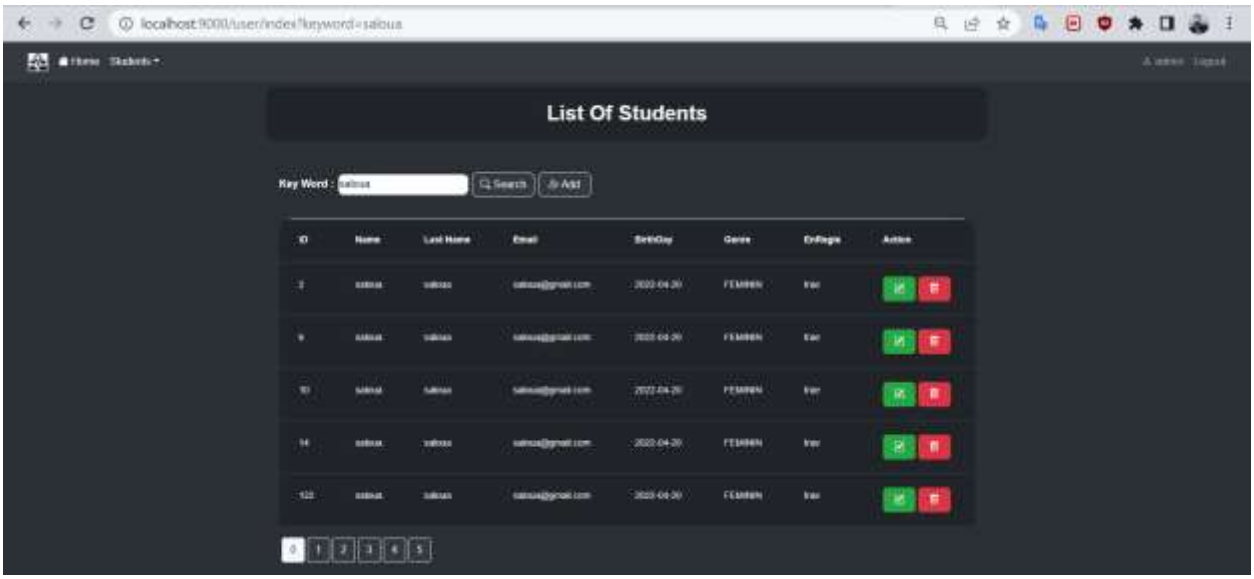


8 Page login (avec le role Admin)

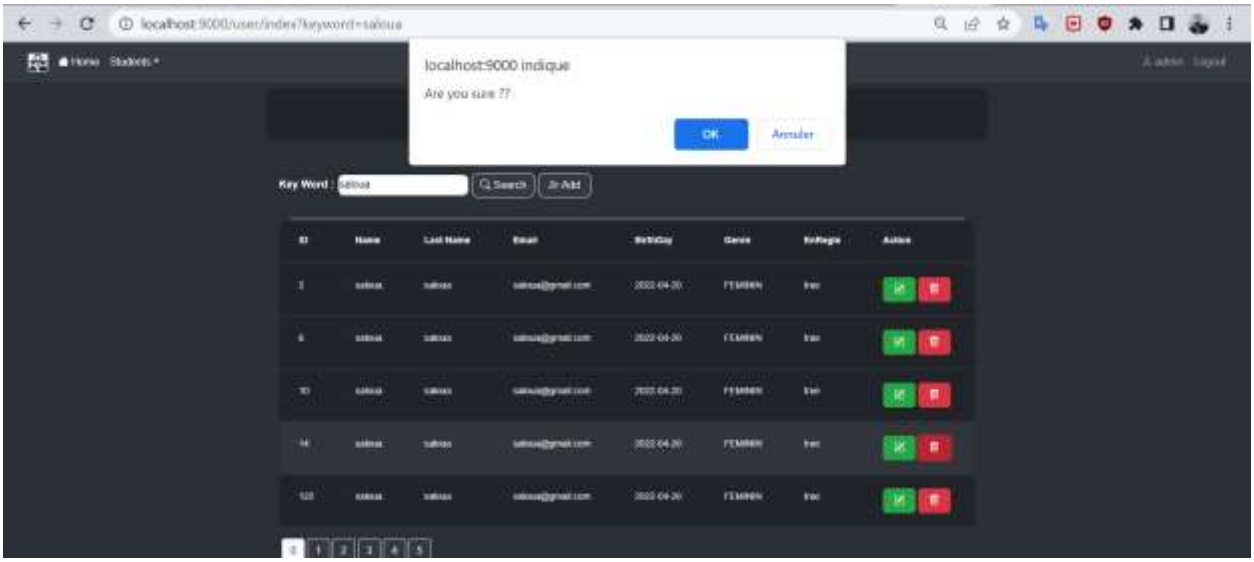
8 Authentification en tant que Admin (rechercher, ajouter, modifie et supprimer)



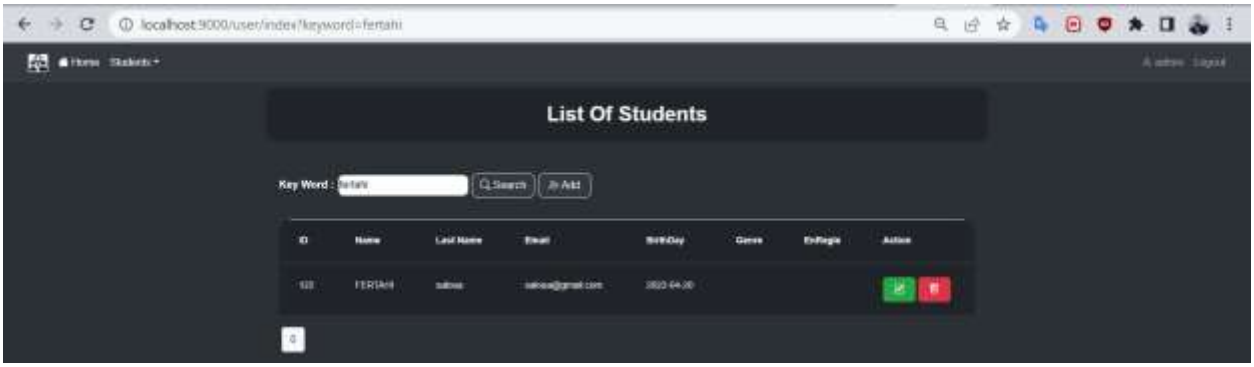
- Rechercher (étudiante Saloua)



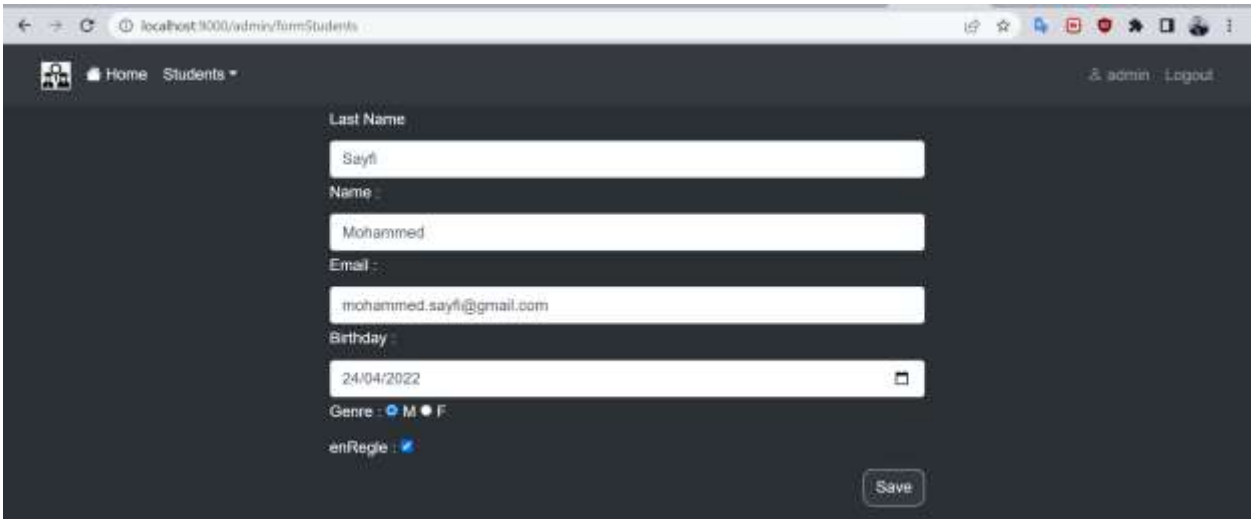
- Supprimer l’étudiante Saloua ayant comme Id 14

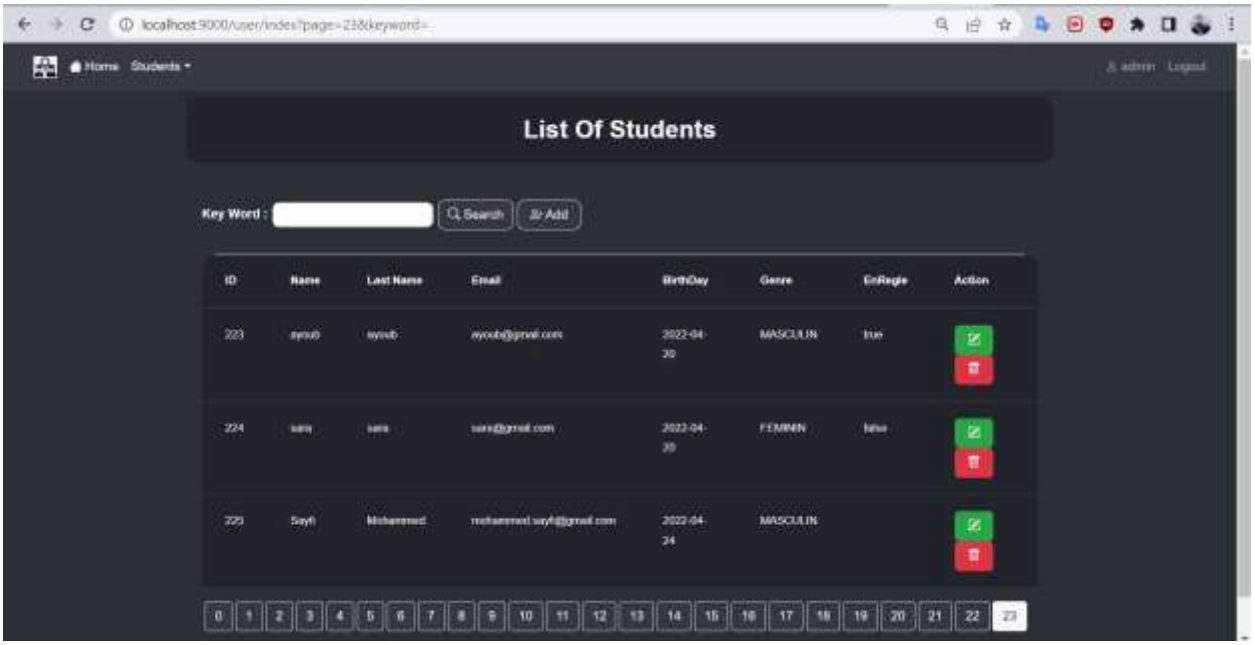


- Modifier l’étudiante Saloua ayant comme Id 122

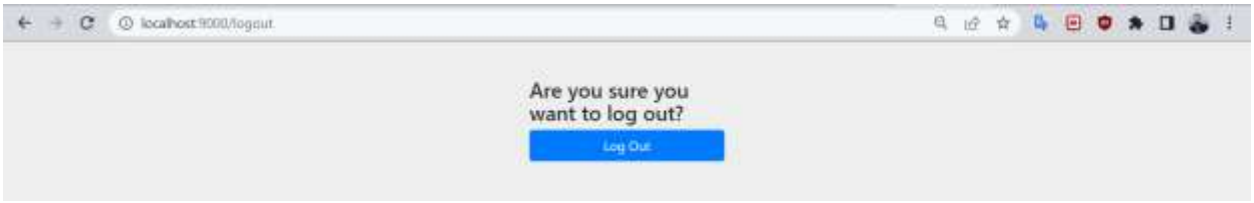


- Ajouter l’étudiant « Mohammed »



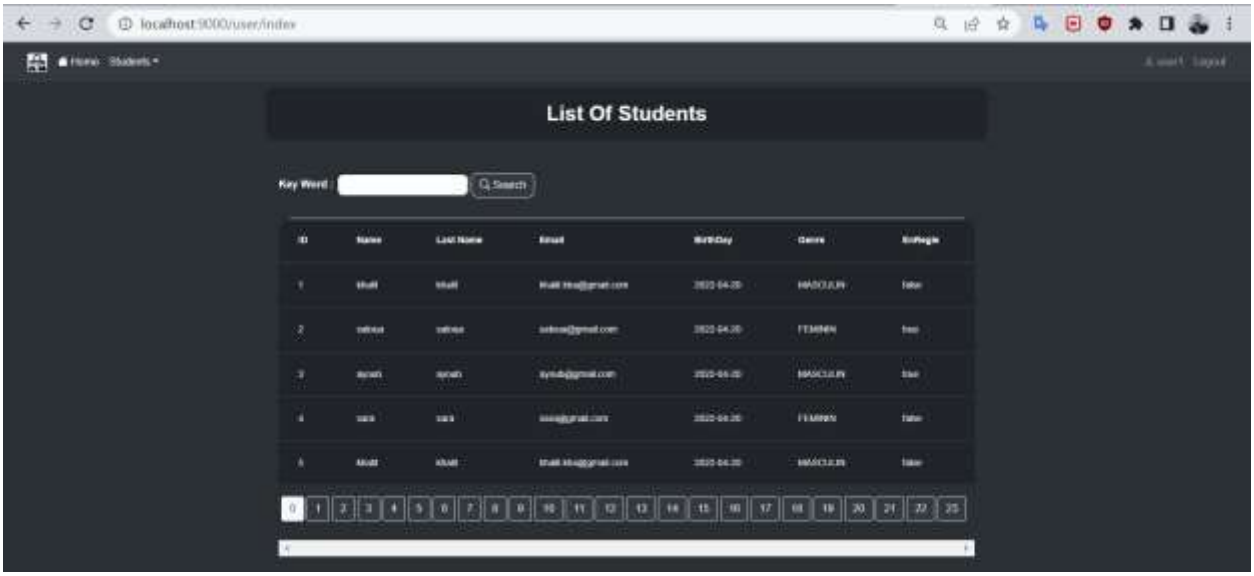


9 Logout

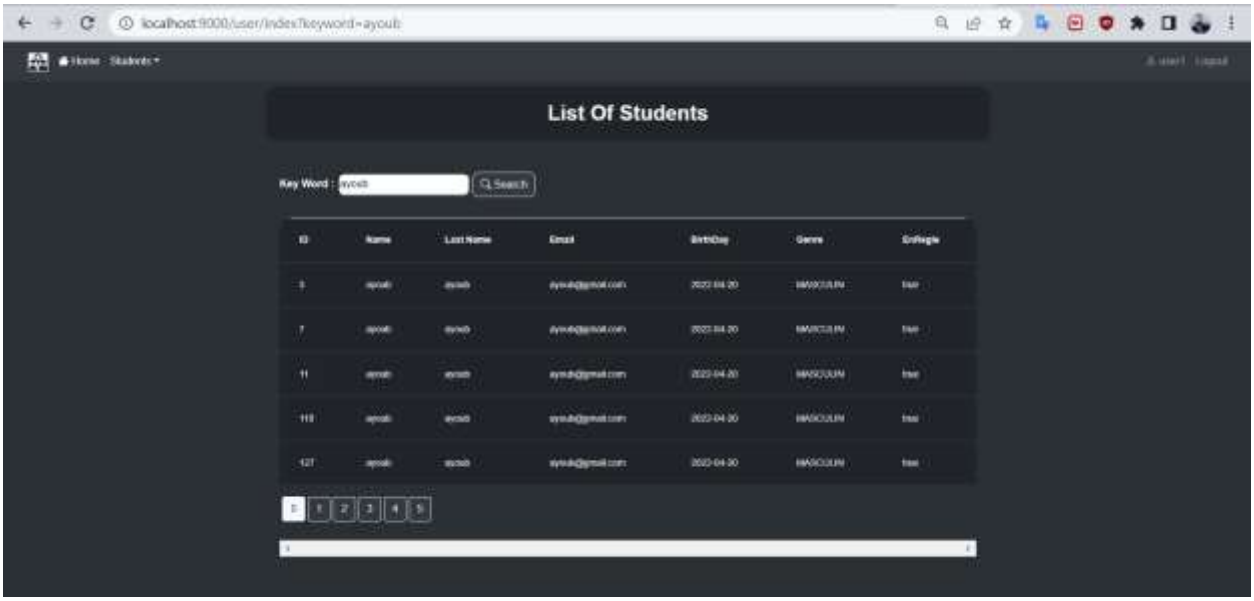


10 Authentification en tant qu'USER

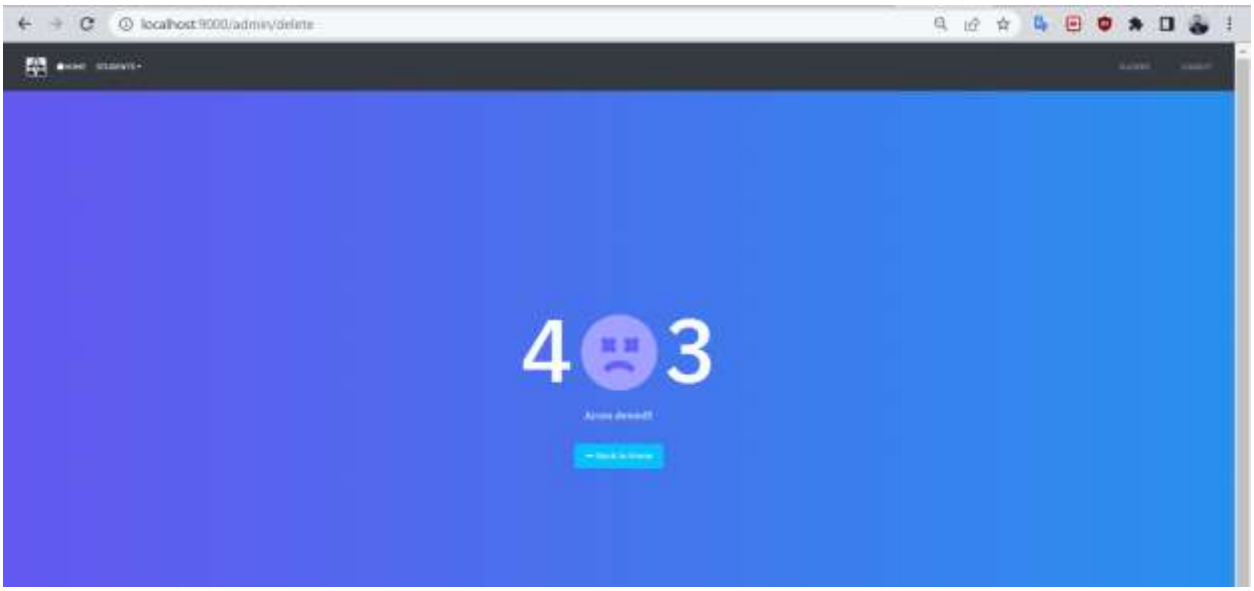
- Consulter la liste des étudiants



- Rechercher un étudiant



- Si l'utilisateur USER1 test l'une des fonctionnalités de l'Administrateur


localhost:9000/admin/delete
403