

## Java - User Story 3 - PDF

- Come Marta
- vorrei poter contare su una funzione di fact-checking
- in modo tale da poter controllare quali notizie pubblicare

### ACCEPTANCE CRITERIA:

- Tre nuovi ruoli: Admin, Revisor, Writer
- Un utente registrato richiede di entrare a far parte del team tramite un form di "lavora con noi"
- Creazione di una dashboard per il proprietario della piattaforma per poter gestire le richieste
- Permettere solo all'admin la modifica e cancellazione delle categorie
- Gli utenti revisori avranno una sezione a loro dedicata con tutti gli articoli da revisionare
- Bottone accetta articolo
- Bottone rifiuta articolo

### Svolgimento ↴

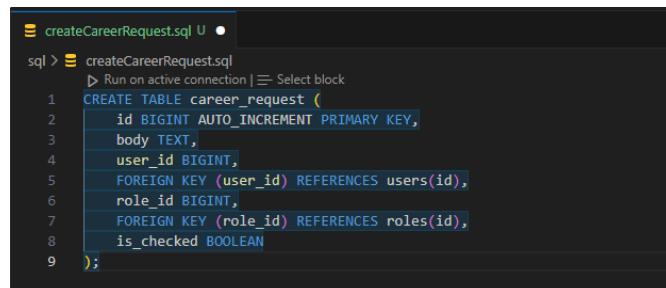
E' arrivato il momento di attivare tutto il meccanismo legato alla richiesta di collaborazione da parte di un utente.

Nel nostro footer abbiamo già inserito il tasto per la richiesta , ma attualmente non ci porta da nessuna parte. Quello che vogliamo è che si apra un form che ci permetta di compilare due campi, il primo e' un piccolo testo all'interno del quale scriviamo la motivazione della nostra richiesta, il secondo è la scelta del ruolo per il quale vogliamo candidarci.

Abbiamo bisogno di iniziare dalla creazione di una nuova tabella all'interno del nostro database.

Creiamo quindi in "sql" un nuovo file "createCareerRequest.sql"

in "sql\createCareerRequest.sql"



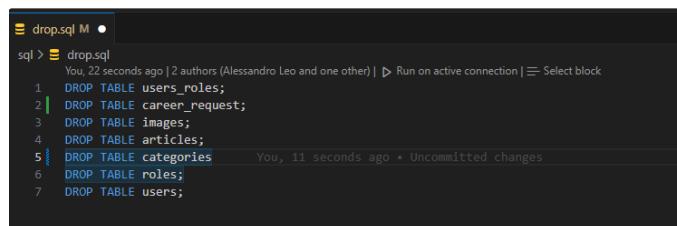
```
createCareerRequest.sql U
sql > createCareerRequest.sql
CREATE TABLE career_request (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    body TEXT,
    user_id BIGINT,
    FOREIGN KEY (user_id) REFERENCES users(id),
    role_id BIGINT,
    FOREIGN KEY (role_id) REFERENCES roles(id),
    is_checked BOOLEAN
);
```

#### E lanciamo lo script sulla nostra connessione al db

Questa tabella ci consentirà di tener traccia delle richieste tramite la relazione utenti/ruoli ed avrà un flag di check per segnalare se la richiesta è stata controllata e accettata o meno.

Fatto questo andiamo di conseguenza a modificare il file "drop.sql"

in "sql\drop.sql"



```
drop.sql M
sql > drop.sql
DROP TABLE users_roles;
DROP TABLE career_request;
DROP TABLE images;
DROP TABLE articles;
DROP TABLE categories;
DROP TABLE roles;
DROP TABLE users;
```

**⚠ ATTENZIONE:** Se lanciamo anche questo script sulla nostra connessione cancelleremo tutti i dati creati e di conseguenza vanno rilanciate nell'ordine tutte le create

### MODELLO(DAO)

Creata la tabella abbiamo bisogno di creare il nostro modello con il quale gestiremo tutti i dati da e verso il database.

Creiamo quindi in “src\main\java\it\aulab\progetto\_finale\_docente\models” il nuovo file “CareerRequest.java”

in “src\main\java\it\aulab\progetto\_finale\_docente\models\CareerRequest.java”

```
src > main > java > it > aulab > progetto_finale_demo_doc > models > CareerRequest.java > CareerRequest > user
1 package it.aulab.progetto_finale_demo_doc.models;
2
3 import jakarta.persistence.Column;
4 import jakarta.persistence.Entity;
5 import jakarta.persistence.GeneratedValue;
6 import jakarta.persistence.GenerationType;
7 import jakarta.persistence.Id;
8 import jakarta.persistence.JoinColumn;
9 import jakarta.persistence.OneToOne;
10 import jakarta.persistence.Table;
11 import lombok.Getter;
12 import lombok.Setter;
13 import lombok.NoArgsConstructor;
14
15 @Setter
16 @Getter
17 @NoArgsConstructor
18 @Entity
19 @Table(name = "career_request")
20 public class CareerRequest {
21     @Id
22     @GeneratedValue(strategy = GenerationType.IDENTITY)
23     private long id;
24     @Column(nullable = false, length = 1000)
25     private String body;
26     @Column
27     private Boolean isChecked;
28
29     @OneToOne
30     @JoinColumn(name = "user_id")
31     private User user;
32
33     @OneToOne
34     @JoinColumn(name = "role_id")
35     private Role role;
36 }
37 }
```

All'interno del quale abbiamo inserito anche le relazioni con la tabella ruoli e la tabella user

## OPERATION CONTROLLER

Procediamo adesso con il controller per gli handler dedicati a questa operazione di richiesta.

Creiamo in “src\main\java\it\aulab\progetto\_finale\_docente\controllers” il file “OperationController.java”

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\OperationController.java”

```
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > OperationController.java > OperationController
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.web.bind.annotation.GetMapping;
9 import org.springframework.web.bind.annotation.RequestMapping;
10
11 import it.aulab.progetto_finale_demo_doc.models.CareerRequest;
12 import it.aulab.progetto_finale_demo_doc.models.Role;
13 import it.aulab.progetto_finale_demo_doc.repositories.RoleRepository;
14
15
16 @Controller
17 @RequestMapping("/operations")
18 public class OperationController {
19
20     @Autowired
21     private RoleRepository roleRepository;
22
23     //Rotta per la creazione di una richiesta di collaborazione
24     @GetMapping("career/request")
25     public String careerRequestCreate(Model viewModel) {
26         viewModel.addAttribute("title", "Inserisci la tua richiesta");
27         viewModel.addAttribute("careerRequest", new CareerRequest());
28
29         List<Role> roles = roleRepository.findAll();
30         //Elimino la possibilità di scegliere il ruolo user nella select del form
31         roles.removeIf(e -> e.getName().equals(anObject("ROLE_USER")));
32         viewModel.addAttribute("roles", roles);
33
34     return "career/requestForm";
35 }
36 }
```

Ed importiamo quello che ci serve (**NB gli import di list e Model**)

## FORM TEMPLATE

Creiamo allora il template che questo handler restituisce in “src\main\resources\templates” creiamo il folder “career” con all'interno il file “requestForm.html”

in “src\main\resources\templates\career\requestForm.html”

```

requestForm.html x
src > main > resources > templates > career > requestForm.html > html > body > head
1   <!DOCTYPE html>
2   <html xmlns="http://www.thymeleaf.org">
3     head th:insert="~{index :: header}"></head>
4   <body>
5     head th:insert="~{index :: navbar}"></head>
6
7
8     <div class="vh-100">
9       <div class="container">
10         <div class="row">
11           <div class="col-12 my-5">
12             <form th:action="@{/operations/career/request/save}" method="POST" th:object="${careerRequest}" enctype="multipart/form-data">
13               <div class="mb-3">
14                 <label for="body" class="form-label">Motivo della richiesta:</label>
15                 <textarea id="body" class="form-control" th:field="*(body)" placeholder="Inserisci motivazione...."></textarea>
16               </div>
17
18               <div class="mb-3">
19                 <label for="role" class="form-label">Per quale ruolo?</label>
20                 <select id="role" th:field="(role)" class="form-select">
21                   <option th:each="role:${roles}" th:value="${role.id}" th:text="${#strings.substring(role.name, 5)}"></option>
22                 </select>
23               </div>
24
25               <button type="submit" class="btn btn-success">Effettua richiesta</button>
26             </form>
27           </div>
28         </div>
29       </div>
30     </div>
31
32     <head th:insert="~{index :: footer}"></head>
33     <script th:replace="~{index :: bootstrapScript}"></script>
34   </body>
35 </html>

```

Fatto questo possiamo procedere con un test di visualizzazione del template appena creato inserendo l'url  
[“http://localhost:8080/operations/career/request”](http://localhost:8080/operations/career/request)

**⚠ NB:** Sempre con un utente loggato o registrato. Logica corretta poichè potremo chiedere di collaborare solo dopo essere utenti effettivamente registrati

Aggiaciamo quindi l'uri al tasto di richiesta nel footer

in “src\main\resources\templates\index.html”

```

index.html M x
src > main > resources > templates > index.html > html > body > footer.bg-dark.text-white.pt-4 > div.container > div.row.justify-content-between > div.col-md-4.mb-3.text-end > a.btn.btn-primary
15   <body>
16
17     <!-- Footer -->
18     <footer th:fragment="footer" class="bg-dark text-white pt-4">
19       <div class="container">
20         <div class="row justify-content-between">
21           <div class="col-4 mb-3">
22             <a href="#"><h5>Aulab IT</h5></a>
23           </div>
24           <div class="col-4 mb-3 text-end">
25             <h5>Lavora con noi!</h5>
26             <a href="#"><span>Richiedi ora!</span></a> 3 seconds ago + (uncommitted changes)
27           </div>
28         </div>
29       </div>
30     </footer>
31
32     <!-- Fine del footer -->
33
34     <!-- fragment script bootstrap -->
35     <script th:fragment="bootstrapScript" src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
36       integrity="sha384-Yvz8YvpnF0tY3Ib60Nkmc5s9rDVLESaA55M0Dz0hy9GKc1ds1k1eN7W6JleHz" crossorigin="anonymous"></script>
37
38     <!-- fine fragment script bootstrap -->
39   </body>
40 </html>

```

Facciamo un test e vediamo che effettivamente ci porta al nostro form di richiesta.

## STORE DELLA RICHIESTA

Adesso procediamo con la creazione dell'handler post che memorizzerà la nuova richiesta.

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\OperationController.java”

```

  OperationController.java 5 •
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > OperationController.java > ...
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.security.Principal;
4 import java.util.List;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Controller;
8 import org.springframework.ui.Model;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.ModelAttribute;
11 import org.springframework.web.bind.annotation.PathVariable;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.RequestMapping;
14 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
15
16 import it.aulab.progetto_finale_demo_doc.models.CareerRequest;
17 import it.aulab.progetto_finale_demo_doc.models.Role;
18 import it.aulab.progetto_finale_demo_doc.models.User;
19 import it.aulab.progetto_finale_demo_doc.repositories.RoleRepository;
20 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
21
22
23
24     Codeium: Refactor | Explain
25     @Controller
26     @RequestMapping("/operations")
27     public class OperationController {
28
29         @Autowired
30         private RoleRepository roleRepository;
31
32         @Autowired
33         private UserRepository userRepository;
34
35
36         //Nota per la creazione di una richiesta di collaborazione
37         @GetMapping("/career/request")
38         public String careerRequestCreate(Model viewModel) {
39             viewModel.addAttribute("title", "Inserisci la tua richiesta");
40             viewModel.addAttribute("careerRequest", new CareerRequest());
41
42             List<Role> roles = roleRepository.findAll();
43             //Elimino la possibilità di scegliere il ruolo user nella select del form
44             roles.removeIf(e -> e.getRoleName().equals(Role.ROLE_USER));
45             viewModel.addAttribute("roles", roles);
46
47             return "career/requestForm";
48         }
49
50         //Nota per il salvataggio di una richiesta di ruolo
51         @PostMapping("/career/request/save")
52         public String careerRequestStore(@ModelAttribute("careerRequest") CareerRequest careerRequest, Principal principal, RedirectAttributes redirectAttributes) {
53
54             User user = userRepository.findByEmail(principal.getName());
55
56             if(careerRequestService.isRoleAlreadyAssigned(user, careerRequest)){
57                 redirectAttributes.addFlashAttribute("errorMessage", "Sei già assegnato a questo ruolo");
58                 return "redirect:/";
59             }
60
61             careerRequestService.save(careerRequest, user);
62
63             redirectAttributes.addFlashAttribute("successMessage", "Richiesta inviata con successo");
64
65             return "redirect:/";
66         }
67     }

```

In questo metodo abbiamo inserito anche un check sulla richiesta che controlla se la richiesta per quel ruolo è già stata inviata oppure l'utente ha già quel ruolo.

#### AGGIUNTA MESSAGGIO DI ERROR IN HOME

Notiamo che nel caso in cui l'utente ha già il ruolo per cui sta effettuando la richiesta invierà al template di redirect un messaggio di ruolo già assegnato. Aggiungiamo allora il codice per la visualizzazione del “errorMessage” nella nostra home dove stiamo effettuando il redirect

in “src\main\resources\templates\home.html”

```

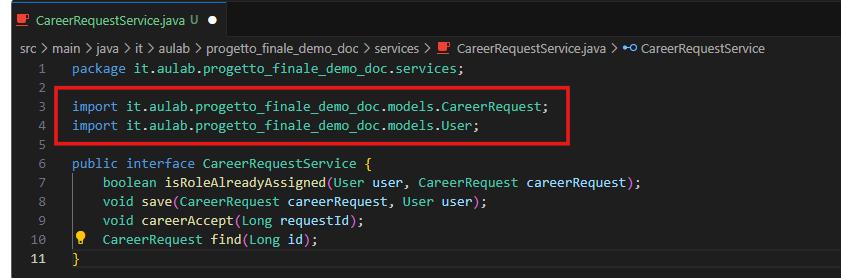
  home.html •
src > main > resources > templates > home.html > home.html > body > div.container
You 6 minutes ago | 2 authors (Alessandro Leo and one other)
1 <!DOCTYPE html>
2 <html lang="en">
3     xmlns:th="http://www.thymeleaf.org"
4     >
5     <head th:insert="~{(index :: head)}>
6     <body>
7
8         <head th:insert="~{(index :: navbar)}>
9
10        <div th:if="${param.notAuthorized}">
11            <div class="alert alert-danger">
12                Not authorized!
13            </div>
14        </div>
15
16        <div th:if="${successMessage}" class="alert alert-success">
17            <p th:text="${successMessage}"></p>
18        </div>
19
20        <div th:if="${errorMessage}" class="alert alert-success">
21            <p th:text="${errorMessage}"></p>
22        </div>
23
24        <div class="container-fluid">
25            <div class="row vh-100 align-items-center">
26                <div class="col-10">
27                    <h1 class="mb-5">Welcome to New Aulab Chronicle</span></h1>
28                </div>
29            </div>
30        </div>

```

Fatto questo notiamo che il metodo precedentemente creato ci segnala degli errori poichè ha bisogno di utilizzare due metodi del "careerRequestService" fino a questo momento non esiste e non possiede ancora i metodi richiesti.

Procediamo con la implementazione creando in "src\main\java\it\aulab\progetto\_finale\_docente\services" il nuovo file "CareerRequestService.java" che corrisponderà all'interfaccia che contiene le firme dei metodi

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestService.java"



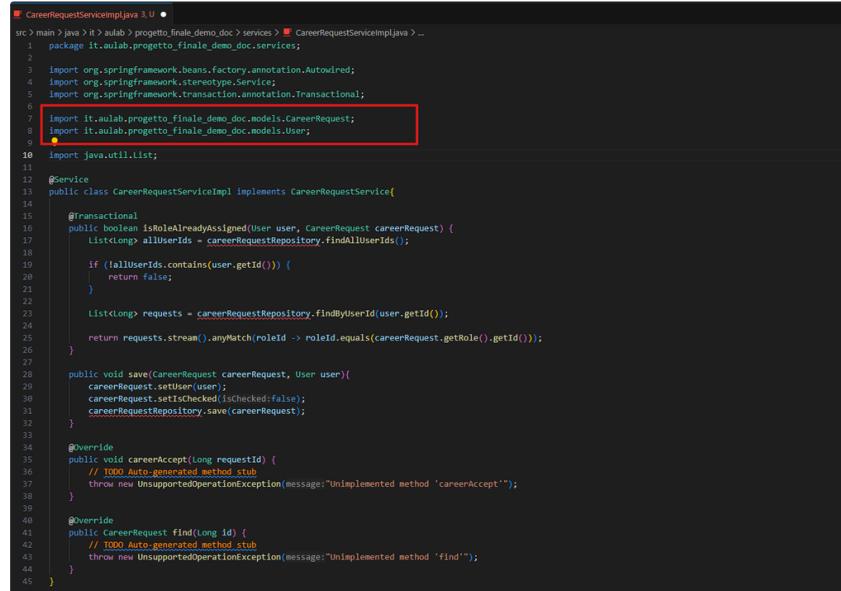
```
1 package it.aulab.progetto_finale_demo.services;
2
3 import it.aulab.progetto_finale_demo.models.CareerRequest;
4 import it.aulab.progetto_finale_demo.models.User;
5
6 public interface CareerRequestService {
7     boolean isRoleAlreadyAssigned(User user, CareerRequest careerRequest);
8     void save(CareerRequest careerRequest, User user);
9     void careerAccept(Long requestId);
10    CareerRequest find(Long id);
11 }
```

ed importiamo quello che serve.

Una volta creata l'interfaccia andiamo con l'implementazione.

creiamo sempre in "src\main\java\it\aulab\progetto\_finale\_docente\services" il nuovo file "CareerRequestServiceImpl.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"



```
1 package it.aulab.progetto_finale_demo.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import it.aulab.progetto_finale_demo.models.CareerRequest;
8 import it.aulab.progetto_finale_demo.models.User;
9
10 import java.util.List;
11
12 @Service
13 public class CareerRequestServiceImpl implements CareerRequestService{
14
15     @Transactional
16     public boolean isRoleAlreadyAssigned(User user, CareerRequest careerRequest) {
17         List<Long> allUserIds = careerRequestRepository.findAllUserIds();
18
19         if (!allUserIds.contains(user.getId())) {
20             return false;
21         }
22
23         List<Long> requests = careerRequestRepository.findByUserId(user.getId());
24
25         return requests.stream().anyMatch(roleId -> roleId.equals(careerRequest.getRole().getId()));
26     }
27
28     public void save(CareerRequest careerRequest, User user){
29         careerRequest.setOwner(user);
30         careerRequest.setChecked(isChecked:false);
31         careerRequestRepository.save(careerRequest);
32     }
33
34     @Override
35     public void careerAccept(Long requestId) {
36         // TODO Auto-generated method stub
37         throw new UnsupportedOperationException("Unimplemented method 'careerAccept'");
38     }
39
40     @Override
41     public CareerRequest find(Long id) {
42         // TODO Auto-generated method stub
43         throw new UnsupportedOperationException("Unimplemented method 'find'");
44     }
45 }
```

ed importiamo quello che serve.

Notiamo immediatamente di aver bisogno di un repository dedicato al "Career".

## REPOSITORY

Creiamo quindi il repository in "src\main\java\it\aulab\progetto\_finale\_docente\repositories" che chiameremo "CareerRequestRepository.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\repositories\CareerRequestRepository.java"

```

1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.CrudRepository;
7 import org.springframework.data.repository.query.Param;
8
9 import it.aulab.progetto_finale_demo_doc.models.CareerRequest;
10
11 public interface CareerRequestRepository extends CrudRepository<CareerRequest, Long>{
12     List<CareerRequest> findByIsCheckedFalse();
13
14     @Query(value = "SELECT user_id FROM users_roles", nativeQuery = true)
15     List<Long> findAllUserIds();
16
17     @Query(value = "SELECT role_id FROM users_roles WHERE user_id = :id ", nativeQuery = true)
18     List<Long> findByUserId(@Param("id") Long id);
19 }

```

ed importiamo quello che serve.

In questa nuova repository abbiamo inserito anche dei metodi che utilizzeremo nel codice più avanti

Creiamo il repository possiamo procedere con l'injection all'interno del service.

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"

```

1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import it.aulab.progetto_finale_demo_doc.models.CareerRequest;
8 import it.aulab.progetto_finale_demo_doc.models.User;
9 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
10
11 import java.util.List;
12
13 @Service
14 public class CareerRequestServiceImpl implements CareerRequestService{
15
16     @Autowired
17     private CareerRequestRepository careerRequestRepository;
18
19     @Transactional
20     public boolean isRoleAlreadyAssigned(User user, CareerRequest careerRequest) {
21         List<Long> allUserIds = careerRequestRepository.findAllUserIds();
22     }

```

Torniamo adesso nel nostro controller dove possiamo fare l'injection del service appena creato

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\OperationController.java"

```

1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import it.aulab.progetto_finale_demo_doc.models.User;
4 import it.aulab.progetto_finale_demo_doc.repositories.RoleRepository;
5 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
6 import it.aulab.progetto_finale_demo_doc.services.CareerRequestService;
7
8
9 @Controller
10 @RequestMapping("/operations")
11 public class OperationController {
12
13     @Autowired
14     private RoleRepository roleRepository;
15
16     @Autowired
17     private UserRepository userRepository;
18
19     @Autowired
20     private CareerRequestService careerRequestService;
21
22
23     //Rotta per la creazione di una richiesta di collaborazione
24     @GetMapping("career/request")
25     public String careerRequestCreate(Model viewModel) {
26         viewModel.addAttribute("userList", userRepository.findAll());
27         return "career-request";
28     }
29
30 }

```

Possiamo ora far un test di richiesta e vedremo che verrà correttamente salvata all'interno del database la nuova richiesta all'interno della tabella "career\_request".

## MAIL VERSO L'ADMIN

Aggiungiamo una cosa molto carina, inviare una mail all'admin sulla avvenuta richiesta da parte di un utente

Andiamo quindi in "CareerRequestServiceImpl.java" e modifichiamo la funzione save().

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"

```

src > main > java > it > aulab > progetto_finale_demo_doc > services > CareerRequestServiceImpl.java > CareerRequestServiceImpl
14 public class CareerRequestServiceImpl implements CareerRequestService {
15     ...
16     ...
17     ...
18     ...
19     ...
20     ...
21     ...
22     ...
23     ...
24     ...
25     ...
26     ...
27     ...
28     ...
29     ...
30     ...
31     ...
32     ...
33     ...
34     ...
35     ...
36     ...
37     public void save(CareerRequest careerRequest, User user) {
38         careerRequest.setUser(user);
39         careerRequest.setIsChecked(false);
40         careerRequestRepository.save(careerRequest);
41         ...
42         ...
43         ...
44         ...
45         ...
46         ...
47         ...
48         ...
49         ...
50         ...
51     }
52 }

```

Ma notiamo immediatamente di aver bisogno di un service per le email che non abbiamo, quindi dobbiamo crearlo

## EMAIL SERVICE

Creiamo quindi in "src\main\java\it\aulab\progetto\_finale\_docente\services" la nuova interfaccia che andremo poi ad implementare che chiameremo "EmailService.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\services\EmailService.java"

```

src > main > java > it > aulab > progetto_finale_demo_doc > services > EmailService.java > EmailService > sendSimpleEmail(String, String, String)
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 public interface EmailService {
4     void sendSimpleEmail(String to, String subject, String text);
5 }

```

Creata l'interfaccia dedichiamoci all'implementazione, creiamo quindi in "src\main\java\it\aulab\progetto\_finale\_docente\services" il nuovo file "EmailServiceImpl.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\services\EmailServiceImpl.java"

```

src > main > java > it > aulab > progetto_finale_demo_doc > services > EmailServiceImpl.java > EmailServiceImpl
1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.mail.SimpleMailMessage;
5 import org.springframework.mail.javamail.JavaMailSender;
6 import org.springframework.scheduling.annotation.Async;
7 import org.springframework.stereotype.Service;
8
9 @Service
10 public class EmailServiceImpl implements EmailService{
11
12     @Autowired
13     private JavaMailSender mailSender;
14
15     @Async
16     public void sendSimpleEmail(String to, String subject, String text) {
17         SimpleMailMessage message = new SimpleMailMessage();
18         message.setFrom("aulabpost@administration.com");
19         message.setTo(to);
20         message.setSubject(subject);
21         message.setText(text);
22         mailSender.send(message);
23     }
24 }

```

Dove all'interno di questo service sfruttiamo il bean appartenente allo spring framework che si occuperà dell'invio della mail.

Creati interfaccia ed implementazione possiamo effettuare l'injection all'interno di "CareerRequestServiceImpl.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"

```

1 package it.aulab.progetto_finale_demo.services;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 import it.aulab.progetto_finale_demo.models.CareerRequest;
8 import it.aulab.progetto_finale_demo.models.User;
9 import it.aulab.progetto_finale_demo.repositories.CareerRequestRepository;
10
11 import java.util.List;
12
13 @Service
14 public class CareerRequestServiceImpl implements CareerRequestService{
15
16     @Autowired
17     private CareerRequestRepository careerRequestRepository;
18
19     @Autowired
20     private EmailService emailService;
21
22     @Transactional
23     public boolean isRoleAlreadyAssigned(User user, CareerRequest careerRequest) {
24         List<Long> allUserIds = careerRequestRepository.findAllUserIds();
25

```

Ultimo step prima di poter fare un test è inserire le configurazioni del nostro provider di mail all'interno dell'application properties.

Utilizzeremo uno strumento per testare l'invio di email in ambienti di sviluppo, intercettando i messaggi senza farli arrivare a destinatari reali, Mailtrap.

**⚠ NB:** Se non si conosce mailtrap fare riferimento alla documentazione relativa.

Su mailtrap non esiste una configurazione creata appositamente per spring ma semplicemente possiamo utilizzare i driver recuperandoli dalla configurazione per Laravel.

in "src\main\resources\application.properties" **aggiungiamo**

```

1 spring.mail.host=<Inseriamo il valore di MAIL_HOST>
2 spring.mail.port=<Inseriamo il valore di MAIL_PORT>
3 spring.mail.username=<Inseriamo il valore di MAIL_USERNAME>
4 spring.mail.password=<Inseriamo il valore di MAIL_PASSWORD>
5 spring.mail.properties.mail.smtp.auth=true
6 spring.mail.properties.mail.smtp.starttls.enable=true
7 spring.mail.properties.mail.smtp.starttls.required=true

```

**💡 Inseriamo tutte le configurazioni così come sopra modificando solo host, port, username, password**

Inserita la nuova configurazione possiamo effettuare un test con una richiesta per un ruolo e vedremo che oltre al salvataggio in DB verrà anche inviata la mail su mailtrap.

**⚠ IMPORTANTE:** Nota sui ruoli. Se si assegna un ruolo manualmente tramite DB ad un utente, o si effettua qualsiasi operazione su un utente per cambiargli il ruolo, se l'utente è loggato non basterà ricaricare la pagina per assegnare i nuovi permessi, ma OGNI VOLTA che un utente riceve un cambio ruolo (o una aggiunta) bisogna sloggarsi e riloggarsi così che vengano assegnate le authorities.

## DASHBOARD ADMIN

Una volta che la richiesta arriva correttamente gestiamo tutte le richieste inviate, per poterlo fare avremo bisogno di una dashboard nella quale solo l'admin può accedere.

Iniziamo con la creazione dell'handler all'interno del nostro "UserController" all'interno di "src\main\java\it\aulab\progetto\_finale\_docente\controllers" in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java"

```

 UserController.java ●
src > main > java > it.aulab > progetto_finale_demo_doc > controllers > UserController.java > categoryService
16 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
17
18 import it.aulab.progetto_finale_demo_doc.services.ArticleService;
19 import it.aulab.progetto_finale_demo_doc.services.CategoryService;
20 import it.aulab.progetto_finale_demo_doc.services.UserService;
21 import jakarta.servlet.http.HttpServletRequest;
22 import jakarta.servlet.http.HttpServletResponse;
23 import jakarta.validation.Valid;
24
25
26 import it.aulab.progetto_finale_demo_doc.models.User;
27 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
28 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
29 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
30
31 You, 5 days ago | 2 authors (Alessandro Leo and one other)
32 @Controller
33 public class UserController {
34
35     @Autowired
36     private UserService userService;
37
38     @Autowired
39     private ArticleService articleService;
40
41     @Autowired
42     private CareerRequestRepository careerRequestRepository;
43
44     @Autowired
45     private CategoryService categoryService; You, 5 days ago * (1 committed changes)

101 //Rotta per la ricerca degli articoli in base all'utente
102 @GetMapping("search/{id}")
103 public String userArticlesSearch(@PathVariable("id") long id, Model viewModel) {
104     User user = userService.findById(id);
105     viewModel.addAttribute("title", "Tutti gli articoli trovati per utente " + user.getUsername());
106
107     List<ArticleDto> articles = articleService.searchByAuthor(user);
108     viewModel.addAttribute("articles", articles);
109
110     return "article/articles";
111 }
112
113 //Rotta per la dashboard dell'admin
114 @GetMapping("admin/dashboard")
115 public String adminDashboard(Model viewModel) {
116     viewModel.addAttribute("title", "Richieste ricevute");
117     viewModel.addAttribute("requests", careerRequestRepository.findByIsCheckedFalse());
118     viewModel.addAttribute("categories", categoryService.readAll());
119
120     return "admin/dashboard";
121 }
122
123 }

```

Dove, come possiamo vedere, l'utilizzo delle derived query che abbiamo inserito in precedenza all'interno del nostro CareerRequestRepository "findByIsCheckedFalse" , che filtra tutti i record eliminando quelli che hanno il campo is\_checked a true.

Costruiamo adesso il template "dashboard" all'interno del folder "admin" che creeremo all'interno di "src\main\resources\templates" non essendo stato ancora creato.

in "src\main\resources\templates\admin\dashboard.html"

```

dashboard.html U ●
src > main > resources > templates > admin > dashboard.html > ...
1 <!DOCTYPE html>
2 <html xmlns="http://www.thymeleaf.org">
3     <head th:insert="~{index :: head}"></head>
4     <body>
5         <head th:insert="~{index :: navbar}"></head>
6
7         <div th:if="${successMessage}" class="alert alert-success">
8             <p th:text="${successMessage}"></p>
9         </div>
10
11         <div class="container min-vh-100">
12             <div class="row">
13                 <div class="col-12 my-5 d-flex justify-content-center align-items-center">
14                     <span class="h1">Dashboard Admin</span>
15                 </div>
16
17                 <div class="col-12 my-5 d-flex justify-content-center align-items-center">
18                     <span class="h2">Richieste ricevute</span>
19                 </div>
20
21                 <div class="col-12">
22                     <table class="table table-striped table-responsive-lg">
23                         <thead>
24                             <tr>
25                                 <th>Id</th>
26                                 <th>Utente</th>
27                                 <th>Per il ruolo</th>
28                                 <th>#Actions</th>
29                             </tr>
30                         </thead>
31                         <tbody>
32                             <tr th:each="request : ${requests}">
33                                 <td>${request.id}</td>
34                                 <td>${request.user.username}</td>
35                                 <td>${request.role.name}</td>
36                                 <td>
37                                     <a href="#"(operation/career/request/detail/{id}(id=${request.id})) class="btn btn-primary w-2">
38                                         <i class="fa-solid fa-bars"></i>
39                                     </a>
40                                 </td>
41                             </tr>
42                         </tbody>
43                     </table>
44                 </div>
45             </div>
46         </div>
47
48         <head th:insert="~{index :: footer}"></head>
49         <script th:replace="~{index :: bootstrapScript}"></script>
50
51     </body>
52 </html>

```

L'idea qual è?

Di mostrare in una tabella tutte le richieste con un tasto , questo consente all'admin di aprire un dettaglio all'interno del quale potrà leggere la richiesta ed eventualmente accettarla.

## DETTAGLIO RICHIESTA

Costruiamo il dettaglio della richiesta.

Partendo sempre dal controller andiamo all'interno del nostro "OperationController" ed aggiungiamo l'handler della richiesta in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\OperationController.java"

```
OperationController.java U
...
66     /**
67      * @GetMapping("career/request/detail/{id}")
68      * public String careerRequestDetail(@PathVariable("id") Long id, Model viewModel) {
69      *     viewModel.addAttribute("title", "Dettaglio richiesta");
70      *     viewModel.addAttribute("request", careerRequestService.find(id));
71      *     return "career/requestDetail";
72      * }
```

Ma l'handler appena inserito utilizza il metodo find() del "CareerRequestService" che non è stato ancora implementato con la giusta logica, modifichiamolo.

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"

```
CareerRequestServiceImpl.java U
...
48     @Override
49     public CareerRequest find(long id) {
50         return careerRequestRepository.findById(id).get();
51     }
```

Dedichiamoci al template che verrà richiamato dall'handler "careerRequestDetail", all'interno del folder "src\main\resources\templates\career" creiamo il nuovo template "requestDetail.html".

in "src\main\resources\templates\career\requestDetail.html"

```
requestDetail.html U
...
45     <head th:insert="~{index :: footer}"></head>
46     <script th:replace="~{index :: bootstrapScript}"></script>
47   </body>
48 </html>
```

Il dettaglio che verrà mostrato conterrà due tasti uno dei quali lo utilizzeremo per accettare la richiesta e come vediamo nel "th:action" punta ad un uri che attualmente non corrisponde ad alcun handler.

Andiamo allora nel controller "OperationController" ed implementiamolo.

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\OperationController.java"

```

1  OperationController.java
2  src > main > java > it > aulab > progetto_finale_demo_doc > controllers > OperationController.java
3  public class OperationController {
4      ...
5      @PostMapping("career/request/detail/{id}")
6      public String careerRequestDetail(@PathVariable("id") long id, Model viewModel) {
7          viewModel.addAttribute("attributeName","title", attributeValue:"Dettaglio richiesta");
8          viewModel.addAttribute("attributeName","request", careerRequestService.findById(id));
9          return "career/requestDetail";
10     }
11
12     //Rotta per l'accettazione di una richiesta
13     @PostMapping("career/request/accept/{requestId}")
14     public String careerRequestAccept(@PathVariable long requestId, RedirectAttributes redirectAttributes) {
15         careerRequestService.careerAccept(requestId);
16         redirectAttributes.addAttribute("attributeName","successMessage", attributeValue:"Ruolo abilitato per l'utente");
17         return "redirect:/admin/dashboard";
18     }
19
20 }

```

Ma questo handler utilizza il metodo "careerAccept" del "CareerRequestService" che esiste ma non è stato implementato con la giusta logica.

Andiamo allora nel nostro service

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CareerRequestServiceImpl.java"

```

1  CareerRequestServiceImpl.java
2  src > main > java > it > aulab > progetto_finale_demo_doc > services > CareerRequestServiceImpl.java
3  package it.aulab.progetto_finale_demo_doc.services;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.stereotype.Service;
7  import org.springframework.transaction.annotation.Transactional;
8
9  import it.aulab.progetto_finale_demo_doc.models.CareerRequest;
10 import it.aulab.progetto_finale_demo_doc.models.User;
11 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
12 import it.aulab.progetto_finale_demo_doc.repositories.RoleRepository;
13 import it.aulab.progetto_finale_demo_doc.repositories.UserRepository;
14
15 import java.util.List;
16
17 @Service
18 public class CareerRequestServiceImpl implements CareerRequestService{
19
20     @Autowired
21     private CareerRequestRepository careerRequestRepository;
22
23     @Autowired
24     private EmailService emailService;
25
26     @Autowired
27     private UserRepository userRepository;
28
29     @Autowired
30     private RoleRepository roleRepository;
31
32
33     @Override
34     public void careerAccept(long requestId) {
35         //Recovery la richiesta
36         CareerRequest request = careerRequestRepository.findById(requestId).get();
37
38         //Dalla richiesta estrago l'utente richiedente ed il ruolo richiesto
39         User user = request.getUser();
40         Role role = request.getRole();
41
42         //Recupero tutti i ruoli che l'utente già possiede ed aggiungo quello nuovo
43         List<Role> rolesUser = user.getRoles();
44         Role newRole = roleRepository.findByName(role.getName());
45         rolesUser.add(newRole);
46
47         //Salvo tutte le nuove modifiche
48         user.setRoles(rolesUser);
49         userRepository.save(user);
50         request.setChecked(isChecked=true);
51         careerRequestRepository.save(request);
52
53         emailService.sendSimpleEmail( user.getEmail() , subject:"Ruolo abilitato" ,text:"ciao, la tua richiesta di collaborazione è stata accettata dalla nostra amministrazione");
54     }
55
56     @Override
57     public CareerRequest find(Long id) {
58         return careerRequestRepository.findById(id).get();
59     }
60 }

```

Ed importiamo ciò che ci serve anche con le injection.

Questo nuovo metodo come vediamo salva la richiesta come accettata e invia una mail all'utente interessato confermandogli che la richiesta è stata accettata

## NAVBAR

Ultimo step andiamo a creare il collegamento sulla navbar e questo sarà visibile solo ad utenti che hanno il ruolo di ADMIN

in "src\main\resources\templates\index.html"

```

src > main > resources > templates > index.html > index.html > body > navbar.navbar-expand-lg navbar-light > div.container-fluid > div#navbarSupportedContent.navbar-collapse > ul.navbar.nav-item-auto.mr-2.mb-lg-0 >
You, 1 second ago | 2 authors (Alessandro Leo and one other)
1 <!DOCTYPE html>
2 <html xmlns="http://www.thymeleaf.org">
3
4 <!-- Fragment head -->
5 <head th:fragment="head">
6   <meta charset="UTF-8" />
7   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QwTKyjpEJSvNiaRU0F0ePoKyCtnYmSpklyT2b4jxh3MbjV6Hw+ALwIH" crossorigin="anonymous" />
9
10  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-S9tSMc+bZxg9644uIX+LJA29/29PKIZQIA6Ta86wefbzKcmRfYX3d9nCVQg9JkQnSC3M6A--" crossorigin="anonymous" referrerpolicy="no-referrer" />
11
12  <title th:text="${title}">Title</title>
13 </head>
14 <!-- Fine fragment head -->
15 <div th:fragment="navBar">
16   <!-- Inizio navbar -->
17   <nav th:fragment="navBar" class="navbar navbar-expand-lg navbar-light bg-light">
18     <div class="container-fluid">
19       <a class="navbar-brand" href="#">Aulab Chronicle</a>
20       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
21         <span class="navbar-toggler-icon"></span>
22       </button>
23       <div class="collapse navbar-collapse" id="navbarSupportedContent">
24         <ul class="navbar-nav me-auto mb-2 mb-lg-0" sec:authorize="isAuthenticated">
25           <li><a href="#">Articles</a></li>
26           <li><a href="#">Create articolo</a></li>
27           <li><a href="#">Tutti gli articoli</a></li>
28           <li><a href="#">Dashboard admin</a></li>
29           <li><a href="#">Accesso</a></li>
30         </ul>
31       <div class="d-flex" sec:authorize="hasRole('ROLE_ADMIN')">
32         <span>Alessandro Leo, last month × (6) completed</span>
33       </div>
34     </div>
35   </nav>
36 </div>
37

```

Possiamo ora fare un test e vediamo che tutto funziona, anche l'accettazione del ruolo aggiunge un nuovo record nella tabella "users\_role" assegnando all'utente il nuovo ruolo.

## SECURITY

Ma abbiamo un problema di sicurezza. Se provassimo l'uri "/admin/dashboard" quindi effettuiamo una richiesta verso "<http://localhost:8080/admin/dashboard>" pur non essendo amministratori vediamo la dashboard e questo non è corretto.

Dobbiamo quindi aggiornare le nostre configurazioni di sicurezza e fare qualche modifica per visualizzare un feedback visivo.

in "src\main\java\it\aulab\spec\_prog\_finale\config\SecurityConfig.java" modifichiamo

```

src > main > java > it > aulab > progetto_finele_demo_doc > config > SecurityConfig.java > SecurityConfig > filterChain(HttpSecurity)
20 public class SecurityConfig {
21
22     @Bean
23     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
24         http
25             .csrf(csrf -> csrf.disable())
26             .authorizeHttpRequests(authorize) ->
27                 .requestMatchers("/**").permitAll()
28                 .requestMatchers("/articles/**", "/categories/**", "/images/**", "/articles/detail/**", "/categories/search/{id}", "/search/{id}").permitAll()
29                 .anyRequest().authenticated()
30             .formLogin(form ->
31                 form.loginPage("/login")
32                 .loginProcessingUrl("/loginProcessingUrl")
33                 .defaultSuccessUrl("/loginSuccessUrl")
34                 .permitAll()
35             )
36             .logout(logout ->
37                 logout.logoutRequestMatcher(new AntPathRequestMatcher(pattern:"/logout"))
38                 .permitAll()
39             )
40             .exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
41             .sessionManagement(session ->
42                 session.creationPolicy(SessionCreationPolicy.IF_REQUIRED)
43                 .maximumSessions(1)
44                 .expiredUrl("/login?session-expired=true")
45             );
46         }
47         return http.build();
48     }
49 }
50
51
52

```

Dove abbiamo aggiunto un ruolo per la rotta che sarà permessa solo ad utenti con ruolo admin.

Facciamo un nuovo test verso l'uri con un utente che non ha ruolo admin e vedremo che verremo reindirizzati nella home con un falsh message che ci mostra di non essere autorizzati.

Questo meccanismo lo abbiamo implementato già in precedenza con i controlli di sicurezza per gli errori "403" nella US1.

## CAMPANELLA DI NOTIFICA

Aggiungiamo una piccola particolarità. Attualmente non abbiamo un feedback visivo chiaro che avvisa l'admin della presenza di richieste da accettare.

Facciamo quindi in modo che quando ci sono richieste da accettare ci viene mostrata una piccola campanellina con il numero delle richieste da accettare.

i (EXTRA: MIGLIORARE IL SISTEMA DI NOTIFICA SEGNANDO SOLO LE NUOVE RICHIESTE DA VISIONARE, QUELLE RIFIUTATE NON DEVONO RIMANERE IN CONTEGGIO O RENDERE NON CONTEGGIATE SE VISUALIZZATE)

Per poter creare questo sistema dobbiamo creare una implementazione dell'interfaccia "HandlerInterceptor" che ci consente di gestire all'interno di tutti i template dei dati condivisi senza dover necessariamente passare per i controller e passarli ogni volta alle vista. Quindi tutti i template condivideranno questi dati.

Creiamo quindi all'interno di "src\main\java\it\aulab\spec\_prog\_finale\config" una nuova classe "NotificationInterceptor.java" che gestirà questi dati

in "src\main\java\it\aulab\progetto\_finale\_docente\config\NotificationInterceptor.java"

```

NotificationInterceptor.java U X
src > main > java > it > aulab > progetto_finale_demo_doc > config > NotificationInterceptor.java > postHandle(HttpServletRequest, HttpServletResponse, Object, ModelAndView)
1 package it.ulab.progetto_finale_demo_doc.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Component;
5 import org.springframework.web.servlet.HandlerInterceptor;
6 import org.springframework.web.servlet.ModelAndView;
7
8 import it.ulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
9 import jakarta.servlet.http.HttpServletRequest;
10 import jakarta.servlet.http.HttpServletResponse;
11
12
13 @Component
14 public class NotificationInterceptor implements HandlerInterceptor {
15
16     @Autowired
17     CareerRequestRepository careerRequestRepository;
18
19     @Override
20     public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
21             ModelAndView modelAndView) throws Exception {
22         if (modelAndView != null && request.isUserInRole("ROLE_ADMIN")) {
23             int careerCount = careerRequestRepository.findByIsCheckedFalse().size();
24             modelAndView.addObject("careerRequests", careerCount);
25         }
26     }
27 }

```

Creando questo "intercettatore" abbiamo bisogno di configurarlo nel nostro ambiente MVC (Model View Controller) quindi dobbiamo creare una nuova configurazione che chiameremo "WebConfig" e che implementa l'interfaccia "WebMvcConfigurer"

in "src\main\java\it\aulab\progetto\_finale\_docente\config\WebConfig.java"

```

WebConfig.java U X
src > main > java > it > aulab > progetto_finale_demo_doc > config > WebConfig.java > WebConfig
1 package it.ulab.progetto_finale_demo_doc.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
6 import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
7
8 @Configuration
9 public class WebConfig implements WebMvcConfigurer {
10
11     @Autowired
12     private NotificationInterceptor notificationInterceptor;
13
14     @Override
15     public void addInterceptors(InterceptorRegistry registry) {
16         registry.addInterceptor(notificationInterceptor);
17     }
18 }

```

Configurato il tutto possiamo aggiungere la campanellina all'interno della nostra navbar

in "src\main\resources\templates\index.html"

```

index.html ●
src > main > resources > templates > index.html > index.html > index.html > body > nav.navbar.navbar-expand-lg.navbar-light.bg-light > div.container-fluid > div.navbarSupportedContent.navbar-collapse > div.navbar.nav.nav-auto.nav-2.mb-0 > link
1 <!doctype html>
2 <html xmlns="http://www.thymeleaf.org">
3
4 <!-- fragment head -->
5 <head th:fragment="head">
6     <meta charset="UTF-8" />
7     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
8     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QMkZzgjyXQ3wWvC7QhXlQeSCW3M6a--- crossorigin="anonymous" refererrerpolicy="no-referrer" />
9
10    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.2/css/all.min.css" integrity="sha512-SnHwK+QZgRjYQmEJ/ZI5JLqZl2Q9rTfYX3P9X3pWvC7QhXlQeSCW3M6a--- crossorigin="anonymous" refererrerpolicy="no-referrer" />
11
12    <title th:text="${title} title</title>
13 </head>
14 <!-- fine fragment head -->
15 <body>
16     <div class="d-flex flex-column min-vh-100">
17         <nav th:fragment="navbar" class="navbar navbar-expand-lg navbar-light bg-light">
18             <div class="container-fluid">
19                 <a class="nav-brand" href="#" th:text="#{aulab Chronicle}">
20                     <img alt="Aulab Chronicle logo" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation" />
21                     <span class="nav-brand-icon"></span>
22                 </a>
23                 <div class="collapse navbar-collapse" id="navbarSupportedContent">
24                     <div class="nav navbar-nav me-auto mb-2 mb-lg-0">
25                         <ul class="secAuthUserListAuthenticated list-group">
26                             <li class="nav-item">
27                                 <a class="nav-link" href="/articles/create" th:text="#{crea articolo}</a>
28                             </li>
29                             <li class="nav-item">
30                                 <a class="nav-link" href="/articles/tutti gli articoli" th:text="#{tutti gli articoli}</a>
31                             </li>
32                             <li class="nav-item secAuthUserListHasRole[ROLE_ADMIN]">
33                                 <div class="d-flex">
34                                     <a class="nav-link" href="#" data-bs-toggle="dropdown" data-bs-target="#navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false" th:text="#{Dashboard admin}</a>
35                                     <div class="fa fa-caret-down" th:if="#{careerRequests > 0}"></div>
36                                     <span class="fa fa-bell fa-1 pt-2"></span>
37                                     <span class="badge rounded-pill bg-danger text-white px-1" th:text="#{careerRequests}"></span>
38                                 </div>
39                             </li>
40                         </ul>
41                         <li class="nav-item dropdown">
42                             <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false" th:text="#{Accesso}</a>
43                         <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
44                             <li>
45                                 <a class="dropdown-item" href="#" th:text="#{Logout}</a>
46                             </li>
47                         </ul>
48                     </div>
49                 </div>
50             </nav>
51         </div>
52     </div>
53 
```

Facciamo un test e vediamo che tutto funziona, vedremo sempre la notifica finché ci saranno richieste nella dashboard.

## GESTIONE CATEGORIE ↗

Sappiamo che solo il nostro admin può creare, modificare o cancellare una categoria. Andiamo quindi a creare tutto questo nuovo sistema.

Iniziamo con il controller "CategoryController" dove inseriamo l'handler che ci porterà verso il template di creazione della nuova categoria

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\CategoryController.java"

```
CategoryController.java M X
src > main > java > it > aula > progetto_finale_demo > controllers > CategoryController.java > CategoryController
22 public class CategoryController {
23     @GetMapping("/category/{id}")
24     public String categorySearch(@PathVariable("id") Long id, Model viewModel) {
25         CategoryDTO category = categoryService.read(id);
26
27         viewModel.addAttribute(attributeName:"title", "tutti gli articoli trovati per categoria " + category.getNome());
28
29         List<ArticleDTO> articles = articleService.searchByCategory(modelMapper.map(category, destinationType:Category.class));
30         viewModel.addAttribute(attributeName:"articles", articles);
31
32         return "article/articles";
33     }
34
35     //Rotta per la creazione di una categoria
36     @GetMapping("create")
37     public String categoryCreate(Model viewModel) {
38         viewModel.addAttribute(attributeName:"title", attributeValue:"Crea un categoria");
39         viewModel.addAttribute(attributeName:"category", new Category());
40
41         return "category/create";
42     }
43
44 }
45
46
47
48
49
50
51
52
53
54 } Alessandro Leo, 6 days ago • User story 2 completa
```

Fatto questo creiamo il template all'interno del folder "category" in "src\main\resources\templates" che chiameremo proprio "create.html" in "src\main\resources\templates\category\create.html"

```
src > main > resources > templates > category > create.html > html > body > div.vh-100 > div.container > div.row
1   <!doctype html>
2   <html xmlns:th="http://www.thymeleaf.org">
3   <head th:insert="~{[index :: header]}></head>
4   <body>
5     <head th:insert="~{[index :: navbar]}></head>
6
7     <div class="vh-100">
8       <div class="container">
9         <div class="row">
10
11           <div class="col-12 my-5 d-flex justify-content-between align-items-center">
12             <div class="d-flex justify-content-center align-items-center flex-grow-1">
13               <h1 th:text="${title}">Inserisci una categoria</h1>
14             </div>
15           </div>
16
17           <div class="col-12 my-5">
18             <form th:action="@{/categories}" method="POST" th:object="${category}">
19               <div class="row mb-3">
20                 <div class="col">
21                   <label for="name" class="form-label">Nome categoria:</label>
22                   <input id="name" type="text" class="form-control" th:field="*[name]" placeholder="Inserisci il nome della categoria..."/>
23                   <p th:errors = "[name]" class="text-danger" th:if="${#fields.hasErrors('name')}></p>
24                 </div>
25               </div>
26             </form>
27
28             <button type="submit" class="btn btn-success">Crea categoria</button>
29           </div>
30         </div>
31       </div>
32     </div>
33   </div>
34
35   <head th:insert="~{[index :: footer]}></head>
36   <script th:replace="~{[index :: bootstrapScript]}></script>
37
38 </body>
39 </html>
```

Questo form una volta compilato effettuerà una richiesta post verso il nostro controller. Andiamo allora a creare l'handler per questa richiesta in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\CategoryController.java"

```
CategoryController.java •
```

```
src > main > java > it.aulab.progetto_finale_demo_doc > controllers > CategoryController.java ...  
You, 6 minutes ago | 2 authors (Alessandro Leo and one other)  
1 package it.aulab.progetto_finale_demo_doc.controllers;  
2  
3 import java.util.List;  
4  
5  
6 import org.modelmapper.ModelMapper;  
7 import org.springframework.beans.factory.annotation.Autowired;  
8 import org.springframework.stereotype.Controller; Alessandro Leo, 5 days ago + User story 2 completa  
9 import org.springframework.web.bind.annotation.ModelAttribute;  
10 import org.springframework.validation.BindingResult;  
11 import org.springframework.web.bind.annotation.GetMapping;  
12 import org.springframework.web.bind.annotation.ModeAttribute;  
13 import org.springframework.web.bind.annotation.PostMapping;  
14 import org.springframework.web.bind.annotation.RequestMapping;  
15 import org.springframework.web.bind.annotation.RequestMethod;  
16 import org.springframework.web.servlet.mvc.support.RedirectAttributes;  
17  
import it.ulab.progetto_finale_demo_doc.doc.Article;  
import it.ulab.progetto_finale_demo_doc.category.Category;  
import it.ulab.progetto_finale_demo_doc.models.Category;  
import it.ulab.progetto_finale_demo_doc.services.ArticleService;  
import it.ulab.progetto_finale_demo_doc.services.CategoryService;  
23 import jakarta.validation.Valid;  
24  
You, 6 minutes ago | 2 authors (You and one other)  
25 @Controller  
26 @RequestMapping("/categories")  
27 public class CategoryController {  
28  
    @Autowired  
    private ArticleService articleService;  
31  
    @Autowired  
    private CategoryService categoryService;
```

```

40     public String categorySearch(@PathVariable("id") Long id, Model viewModel) {
41   }
42
43   //Rotta per la creazione di una categoria
44   @GetMapping("create")
45   public String categoryCreate(Model viewModel) {
46     viewModel.addAttribute("title", attributeValue:"Crea un categoria");
47     viewModel.addAttribute(attributeName:"category", new Category());
48     return "category/create";
49   }
50
51   //Rotta per la memorizzazione di una categoria
52   @PostMapping
53   public String categoryStore(@Valid @ModelAttribute("category") Category category,
54                               BindingResult result,
55                               RedirectAttributes redirectAttributes,
56                               Model viewModel) {
57
58     //Controllo degli errori con validazioni
59     if (result.hasErrors()) {
60       viewModel.addAttribute(attributeName:"title", attributeValue:"Crea un categoria");
61       viewModel.addAttribute(attributeName:"category", category);
62       return "category/create";
63     }
64
65     categoryService.create(category, principal:null, file:null);
66     redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Categoria aggiunta con successo!");
67
68     return "redirect:/admin/dashboard";
69   }
70
71 }

```

ed importiamo quello che ci serve.

In questo metodo abbiamo inserito anche la gestione degli errori di validazione.

Richiama però il metodo “create” di “CategoryService” che sappiamo non avere la giusta logica, modifichiamo.

in “src\main\java\it\aulab\progetto\_finale\_doc> services> CategoryService.java”

```

CategoryService.java •
src > main > java > it > aulab > progetto_finale_demo_doc > services > CategoryService.java > CategoryService > create(Category, Principal, MultipartFile)
17  public class CategoryService implements CrudService<CategoryDto, Category, Long>{
18    public List<CategoryDto> readAll() {
19      return dtos;
20    }
21
22    @Override
23    public CategoryDto read(long key) {
24      return modelMapper.map(categoryRepository.findById(key), destinationType:CategoryDto.class);
25    }
26
27    @Override
28    public CategoryDto create(Category model, Principal principal, MultipartFile file) {      Alessandro Leo, last week + US01
29      return modelMapper.map(categoryRepository.save(model), destinationType:CategoryDto.class);
30    }
31
32    @Override
33    public CategoryDto update(Long key, Category model, MultipartFile file) {
34      // TODO Auto-generated method stub
35      throw new UnsupportedOperationException(message:"Unimplemented method 'update'");
36    }
37
38    @Override
39    public void delete(Long key) {
40      // TODO Auto-generated method stub
41      throw new UnsupportedOperationException(message:"Unimplemented method 'delete'");
42    }
43
44  }

```

Per il richiamo del form e del meccanismo di memorizzazione pensiamo di inserire nella dasboard dell’admin una tabella con tutte le categorie disponibili ed a fianco del titolo un tasto con un “+” che permette di creare una nuova categoria

Modifichiamo quindi la dashboard dell’admin

in “src\main\resources\templates\admin\dashboard.html”

```

1 src > main > resources > templates > admin > dashboard.html > HTML > body > div.container.min-vh-100 > div.row
2   <div class="container min-vh-100">
3     <div class="row">
4       <div class="col-12">
5         <table border="1">
6           <thead>
7             <tr>
8               <th>Id</th>
9               <th>Nome</th>
10              <th>Actions</th>
11            </tr>
12          <tbody>
13            <tr th:each="category : ${categories}">
14              <td>${category.id}</td>
15              <td>${category.name}</td>
16              <td>
17                <a href="#">Edit</a>
18                <a href="#">Delete</a>
19              </td>
20            </tr>
21          </tbody>
22        </table>
23      </div>
24    </div>
25    <div class="col-12 my-5 d-flex justify-content-center align-items-center">
26      <span>Tutte le categorie:</span>
27      <a href="#">Crea nuova categoria</a>
28    </div>
29  </div>
30  <div class="col-12">
31    <table border="1">
32      <thead>
33        <tr>
34          <th>Id</th>
35          <th>Nome</th>
36          <th>Actions</th>
37        </tr>
38      <tbody>
39        <tr th:each="category : ${categories}">
40          <td>${category.id}</td>
41          <td>${category.name}</td>
42          <td>
43            <a href="#">Edit</a>
44            <a href="#">Delete</a>
45          </td>
46        </tr>
47      </tbody>
48    </table>
49  </div>
50</div>
51</div>
52</div>
53</div>
54</div>
55</div>
56</div>
57</div>
58</div>
59</div>
60</div>
61</div>
62</div>
63</div>
64</div>
65</div>
66</div>
67</div>
68</div>
69</div>
70</div>
71</div>
72</div>
73</div>
74</div>
75</div>
76</div>
77</div>
78</div>
79</div>
80</div>
81</div>
82</div>
83</div>
84</div>
85</div>
86</div>
87</div>

```

Creando il template possiamo fare un test di memorizzazione di una nuova categoria e vedremo che tutto funziona correttamente.

## MODIFICA E CANCELLAZIONE DI CATEGORIE

Occupiamoci ora della modifica e della cancellazione delle categorie.

Partiamo sempre dal nostro controller “CategoryController”

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\CategoryController.java”

```

1 package it.aulab.progetto_finale_docente.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8
9 import it.aulab.progetto_finale_docente.services.CategoryService;
10
11 @Controller
12 public class CategoryController {
13     private final CategoryService categoryService;
14
15     public CategoryController(CategoryService categoryService) {
16         this.categoryService = categoryService;
17     }
18
19     @GetMapping("/category")
20     public String categoryList(Model model) {
21         model.addAttribute("categories", categoryService.readAll());
22         return "category/list";
23     }
24
25     @GetMapping("/category/create")
26     public String categoryCreate(Model model) {
27         model.addAttribute("category", new Category());
28         return "category/form";
29     }
30
31     @PostMapping("/category")
32     public String categorySave(@Valid @ModelAttribute("category") Category category,
33                               Model model) {
34         if (category.getId() == null) {
35             categoryService.create(category);
36             model.addAttribute("successMessage", "Categoria creata con successo!");
37         } else {
38             categoryService.update(category);
39             model.addAttribute("successMessage", "Categoria aggiornata con successo!");
40         }
41         return "redirect:/category";
42     }
43
44     @GetMapping("/category/edit/{id}")
45     public String categoryEdit(@PathVariable("id") Long id, Model viewModel) {
46         viewModel.addAttribute("title", "Modifica categoria");
47         viewModel.addAttribute("category", categoryService.read(id));
48         return "category/form";
49     }
50
51     @PostMapping("/category/{id}")
52     public String categoryUpdate(@PathVariable("id") Long id, @ModelAttribute("category") Category category,
53                                 Model model) {
54         categoryService.update(category);
55         model.addAttribute("successMessage", "Categoria aggiornata con successo!");
56         return "redirect:/category";
57     }
58
59     @GetMapping("/category/delete/{id}")
60     public String categoryDelete(@PathVariable("id") Long id, Model model) {
61         categoryService.delete(id);
62         model.addAttribute("successMessage", "Categoria cancellata con successo!");
63         return "redirect:/category";
64     }
65
66     @GetMapping("/category/{id}/cancel")
67     public String categoryCancel(@PathVariable("id") Long id, Model model) {
68         categoryService.cancel(id);
69         model.addAttribute("successMessage", "Operazione annullata con successo!");
70         return "redirect:/category";
71     }
72
73     @GetMapping("/category/{id}/uncommit")
74     public String categoryUncommit(@PathVariable("id") Long id, Model model) {
75         categoryService.uncommit(id);
76         model.addAttribute("successMessage", "Operazione salvata con successo!");
77         return "redirect:/category";
78     }
79
80     @GetMapping("/category/{id}/refresh")
81     public String categoryRefresh(@PathVariable("id") Long id, Model model) {
82         categoryService.refresh(id);
83         model.addAttribute("successMessage", "Operazione riconfigurata con successo!");
84         return "redirect:/category";
85     }
86
87     @GetMapping("/category/{id}/log")
88     public String categoryLog(@PathVariable("id") Long id, Model model) {
89         model.addAttribute("log", categoryService.log(id));
90         return "category/log";
91     }
92 }

```

Creata l’handler procediamo come sempre con il template “update.html”

in “src\main\resources\templates\category\update.html”

```

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head th:insert="~{index :: head}"></head>
4 <body>
5     <head th:insert="~{index :: navbar}"></head>
6
7     <div class="vh-100">
8         <div class="container">
9             <div class="row">
10
11                 <div class="col-12 my-5 d-flex justify-content-between align-items-center">
12                     <div class="d-flex justify-content-center align-items-center flex-grow-1">
13                         <h1 th:text="${title}">Modifica la categoria</h1>
14                     </div>
15                 </div>
16
17                 <div class="col-12 my-5">
18                     <form th:action="@{/categories/update/{id}}(id=${category.id})" method="post" th:object="${category}">
19                         <div class="row mb-3">
20                             <div class="col">
21                                 <label for="name" class="form-label">Nome categoria</label>
22                                 <input id="name" type="text" class="form-control" th:field="*{name}">
23                                 placeholder="Inserisci il nome...">
24                                 <p th:errors = "{name}" class="text-danger" th:if="#{fields.hasErrors('name')}"></p>
25                             </div>
26                         </div>
27
28                         <button type="submit" class="btn btn-success">Aggiorna categoria</button>
29                     </form>
30                 </div>
31             </div>
32         </div>
33     </div>
34
35
36     <head th:insert="~{index :: footer}"></head>
37     <script th:replace="~{index :: bootstrapScript}"></script>
38 </body>
39 </html>

```

Questo form dopo aver ricevuto eventuali modifiche cercherà una rotta post per la modifica, creiamola nel controller

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\CategoryController.java"

```

1 public class CategoryController {
2     public String categoryStore(@Valid @ModelAttribute("category") Category category,
3
4         redirectAttributes.addAttribute("successMessage", attributeValue:"Categoria aggiunta con successo!");
5
6         return "redirect:/admin/dashboard";
7     }
8
9
10    //Rotta per la modifica di una categoria
11    @GetMapping("/edit/{id}")
12    public String categoryEdit(@PathVariable("id") Long id, Model viewModel) {
13        viewModel.addAttribute(attributeName:"title", attributeValue:"Modifica categoria");
14        viewModel.addAttribute(attributeName:"category", categoryService.read(id));
15        return "category/update";
16    }
17
18    //Rotta per la memorizzazione delle modifiche di una categoria
19    @PostMapping("/update/{id}")
20    public String categoryUpdate(@PathVariable("id") Long id,
21        @Valid @ModelAttribute("category") Category category,
22        BindingResult result,
23        RedirectAttributes redirectAttributes,
24        Model viewModel) {
25
26        //Controllo degli errori con validazioni
27        if (result.hasErrors()) {
28            viewModel.addAttribute(attributeName:"title", attributeValue:"Modifica categoria");
29            viewModel.addAttribute(attributeName:"category", category);
30            return "category/update";
31        }
32
33        categoryService.update(id, category, fileOrNull);
34        redirectAttributes.addAttribute("successMessage", attributeValue:"Categoria modificata con successo!");
35
36        return "redirect:/admin/dashboard";
37    }
38
39    You, 32 seconds ago • Uncommitted changes
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

```

Questo nuovo handler però ha bisogno del metodo "update" di "CategoryService" che non possiede la giusta logica, modifichiamo.

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"

```

1 package it.aulab.progetto_finale_demo_doc.services;
2
3 import java.security.Principal;
4 import java.util.List;
5 import java.util.ArrayList;
6
7 import org.modelmapper.ModelMapper;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.stereotype.Service;
11 import org.springframework.web.multipart.MultipartFile;
12 import org.springframework.web.server.ResponseStatusException;
13
14 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
15 import it.aulab.progetto_finale_demo_doc.models.Category;
16 import it.aulab.progetto_finale_demo_doc.repositories.CategoryRepository;
17
18
19 @Service
20
21     @Autowired
22     private CategoryRepository categoryRepository;
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
39
40
41
42
43
44
45
46
47
48
49
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

```

```

35
36     @Override
37     public CategoryDto read(Long key) {
38         return modelMapper.map(categoryRepository.findById(key), destinationType:CategoryDto.class);
39     }
40
41     @Override
42     public CategoryDto create(Category model, Principal principal, MultipartFile file) {
43         return modelMapper.map(categoryRepository.save(model), destinationType:CategoryDto.class);
44     }
45
46     @Override
47     public CategoryDto update(Long key, Category model, MultipartFile file) {
48         if (categoryRepository.existsById(key)) {
49             model.setId(key);
50             return modelMapper.map(categoryRepository.save(model), destinationType:CategoryDto.class);
51         } else {
52             throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
53         }
54     }
55
56     @Override
57     public void delete(Long key) {
58         // TODO Auto-generated method stub
59         throw new UnsupportedOperationException(message: "Unimplemented method 'delete'");
60     }
61 }

```

Aggiorniamo ora i nuovi sviluppi ad un tasto che metteremo all'interno della tabella delle categorie nella dashboard dell'admin in direzione del campo action

in "src\main\resources\templates\admin\dashboard.html"

```

1 < dashboard.html U x
2 < main > resources > templates > admin > dashboard.html > < body > < div.container.min-vh-100 > < div.row > < div.col-12 > < table.table.table-striped-table-responsive-lg > < body > < tr > < td> < a href="#" class="text-decoration-none text-dark" >
3 < td > < th > # </th >
4 < td > < th > Nome </th >
5 < td > < th > Actions </th >
6 < /tr >
7 < /thead >
8 < tbody >
9 < tr th:each="category : ${categories}">
10 < td th:text="${category.id}"></td>
11 < td th:text="${category.name}"></td>
12 < td class="text-end">
13 < a href="#" class="text-decoration-none text-dark" >
14 < a href="#" class="fa-solid fa-pencil" >
15 < /a >
16 < /td >
17 < /tr >
18 < /tbody >
19 < /table >
20 < /div >
21 < /div >
22 < /div >
23 < /div >
24 < /div >
25 < /div >
26 < /div >
27 < /div >
28 < /div >
29 < /div >
30 < head th:insert="--{index :: footer}"></head>
31 < script th:replace="--{index :: bootstrapScript}"></script>
32 < /body >
33 < /html >

```

Facciamo un test anche della modifica sulla categoria e vedremo che andrà a buon fine.

## DELETE

Procediamo con la cancellazione di una categoria

Si parte sempre dal controller "CategoryController"

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\CategoryController.java"

```

1 < CategoryController.java M x
2 < main > java > it > aulab > progetto_finale_demo_doc > controllers > CategoryController > < CategoryController > < categoryUpdate(@PathVariable("id")Long id, RedirectAttributes redirectAttributes, Model model) >
3 < public String categoryUpdate(@PathVariable("id")Long id, RedirectAttributes redirectAttributes, Model model) {
4     if (categoryService.existsById(id)) {
5         categoryService.update(id, category, file);
6         redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Categoria modificata con successo!");
7         return "redirect:/admin/dashboard";
8     }
9     return "category/update";
10 }
11
12     @GetMapping("delete/{id}")
13     public String categoryDelete(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {
14
15         categoryService.delete(id);
16         redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Categoria cancellata con successo!");
17     }
18 }

```

Anche questo handler richiede l'utilizzo del metodo "delete" di "CategoryService" non ancora implementato con la giusta logica, procediamo.

in "src\main\java\it\aulab\progetto\_finale\_docente\services\CategoryService.java"

```

1 package it.aulab.progetto_finale_demo.services;
2
3 import java.security.Principal;
4 import java.util.List;
5 import java.util.ArrayList;
6
7 import org.modelmapper.ModelMapper;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.http.HttpStatus;
10 import org.springframework.stereotype.Service;
11 import org.springframework.web.multipart.MultipartFile;
12 import org.springframework.web.server.ResponseStatusException;
13
14 import it.aulab.progetto_finale_demo.dtos.CategoryDto;
15 import it.aulab.progetto_finale_demo.models.Article;
16 import it.aulab.progetto_finale_demo.models.Category;
17 import it.aulab.progetto_finale_demo.repositories.CategoryRepository;
18 import jakarta.transaction.Transactional;
19
20 @Service
21 public class CategoryService implements CrudService<CategoryDto, Category, Long>{
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
}

```

Ed importiamo quello che serve.

Questo metodo “delete” effettua anche un controllo sulla esistenza di possibili relazioni tra la categoria ed un articolo e sgancia la relazione prima di procedere con la cancellazione della categoria stessa, ulteriormente è anche un metodo transazionale poichè verifica in prima istanza che l’operazione sia andata a buon fine per poi salvare tutto definitivamente.

Aggiaciamo questa ultima operazione ad un tasto sempre all’interno della tabella delle categorie nella dashboard dell’admin

in “src\main\resources\templates\admin\dashboard.html”

```

1 <div th:insert="~{index :: footer}"></div>
2 <script th:replace="~{index :: bootstrapScript}"></script>
3 </body>
4 </html>

```

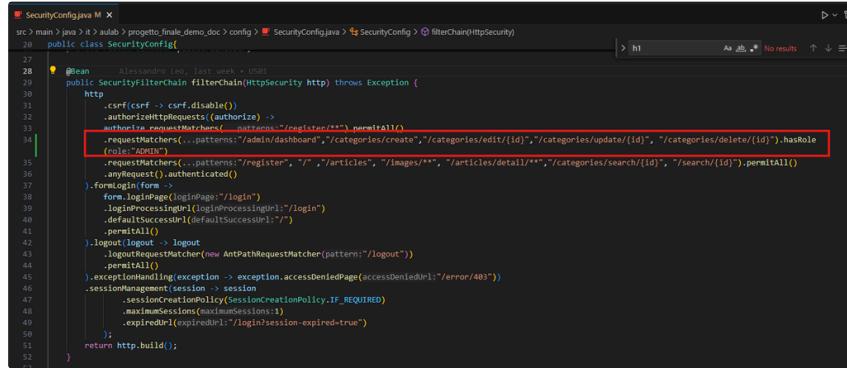
Possiamo effettuare ora un test e vediamo che anche la cancellazione avviene con successo.

## SECURITY

Abbiamo completato tutto il crud per le categorie ma c'è una falla di sicurezza, se gli uri venissero intercettati, anche utenti che non sono admin potrebbero manipolare le categorie.

Andiamo quindi ad applicare un blocco per gli utenti non admin.

in "src\main\java\it\aulab\progetto\_finale\_docente\config\SecurityConfig.java" modifichiamo



```
27     @Bean
28     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
29         http
30             .csrf().disable()
31             .authorizeHttpRequests(authorize) -
32                 .antMatchers("/register/**").permitAll()
33                 .antMatchers("/admin/dashboard", "/categories/create", "/categories/edit/{id}", "/categories/update/{id}", "/categories/delete/{id}").hasRole("ADMIN")
34                 .requestMatchers(patterns:"/articles", "/images/**", "/articles/detail/**", "/categories/search/{id}", "/search/{id}").permitAll()
35             .anyRequest().authenticated()
36             .formLogin(form) -
37                 .loginPage("/login")
38                 .logInSuccessHandler(logInSuccessHandler)
39                 .defaultSuccessUrl(defaultSuccessUrl)
40                 .permitAll()
41             .logout(logout) -
42                 .logoutRequestMatcher(new AntPathRequestMatcher(patterns:"/logout"))
43             .passwordEncoder(passwordEncoder)
44             .exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
45             .sessionManagement(session -> session
46                 .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
47                 .maximumSessions(maximumSessions)
48                 .expiredCookieExpiredUrl("login/session-expired=true")
49             );
50         return http.build();
51     }
52 }
```

Con queste protezioni se provassimo ad usare gli uri vedremmo che ci verranno bloccate.

## GESTIONE REVISORE ↗

Procediamo adesso con la gestione del ruolo del revisore con le modifiche necessarie affinché un articolo vada in revisione dopo essere stato creato e sia visibile solo dopo essere stato accettato dal revisore stesso.

Il primo passo da fare è modificare la tabella "articles" affinché contenga un nuovo campo booleano che chiameremo "is\_accepted"

Creiamo quindi in "sql" un nuovo file chiamato "alterTableArticle.sql"

in "sql\alterTableArticle.sql"

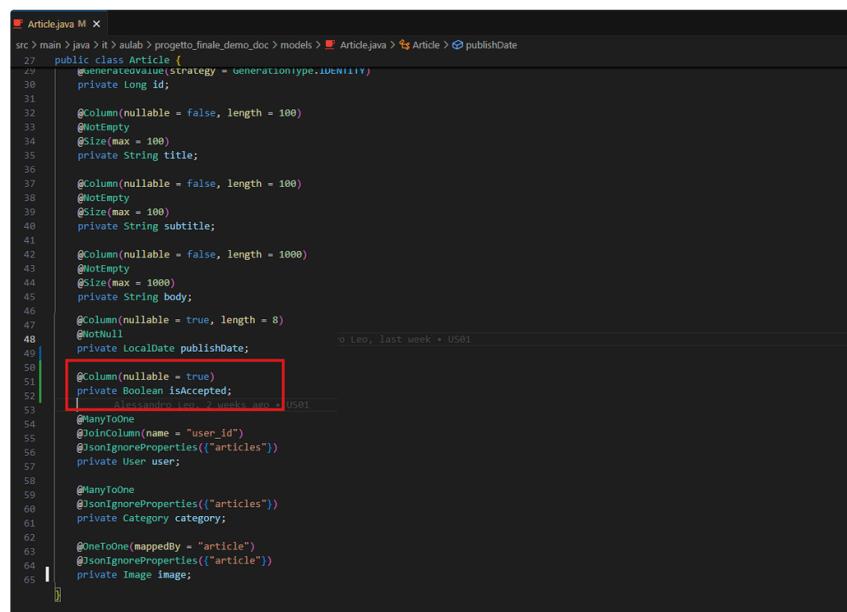
```
1 ALTER TABLE articles
2 ADD COLUMN is_accepted BOOLEAN DEFAULT NULL;
```

E lanciamo lo script sulla connessione

## DTO E DAO

Fatto questo andiamo immediatamente a modificare dao e dto corrispondenti

in "src\main\java\it\aulab\progetto\_finale\_docente\models\Article.java"



```
27     public class Article {
28         @GeneratedValue(strategy = GenerationType.IDENTITY)
29         private Long id;
30
31         @Column(nullable = false, length = 100)
32         @NotEmpty
33         @Size(max = 100)
34         private String title;
35
36         @Column(nullable = false, length = 100)
37         @NotEmpty
38         @Size(max = 100)
39         private String subtitle;
40
41         @Column(nullable = false, length = 1000)
42         @NotEmpty
43         @Size(max = 1000)
44         private String body;
45
46         @Column(nullable = true, length = 8)
47         @NotNull
48         private LocalDate publishDate;
49
50         @Column(nullable = true)
51         private Boolean isAccepted;
52     }
```

E poi il dto

in "src\main\java\it\aulab\progetto\_finale\_docente\dtos\ArticleDto.java"

```

ArticleDto.java •
src > main > java > it > aulab > progetto_finale_demo_doc > dtos > ArticleDto.java > ArticleDto > isAccepted
You, last week | 2 authors (Alessandro Leo and one other)
1 package itaulab.progetto_finale_demo_doc.dtos;
2
3 import java.time.LocalDate;
4
5 import itaulab.progetto_finale_demo_doc.models.Category;
6 import itaulab.progetto_finale_demo_doc.models.Image;
7 import itaulab.progetto_finale_demo_doc.models.User;
8 import lombok.Getter;
9 import lombok.Setter;
10 import lombok.NoArgsConstructor;
11
12 You, last week | 2 authors (Alessandro Leo and one other)
13 @Setter
14 @Getter
15 @NoArgsConstructor
16 public class ArticleDto {
17     private Long id;
18     private String title;
19     private String subtitle;
20     private String body;
21     private LocalDate publishDate;
22     private Boolean isAccepted; You, last week * Uncommitted changes
23     private User user;
24     private Category category;
25     private Image image;
}

```

## MODIFICA DEL SERVICE ARTICLE

Fatte queste modifiche attiviamo il meccanismo affinchè un articolo appena creato dovrà essere prima accettato dal revisore per poter essere effettivamente visualizzato.

Basta quindi far una modifica al nostro “ArticleService” dove inseriamo proprio il set di questo flag a false

in “src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java”

```

ArticleService.java M •
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > create(Article, Principal, MultipartFile)
26 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
27     ...
28
29     @Override
30     public ArticleDto create(Article article, Principal principal, MultipartFile file) {
31         String url = "";
32
33         Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
34         if (authentication != null) {
35             CustomUserDetails userDetailsService = (CustomUserDetails) authentication.getPrincipal();
36             User user = userRepository.findById(userDetailsService.getId()).get();
37             article.setUser(user);
38         }
39
40         if(!file.isEmpty()){
41             try {
42                 CompletableFuture<String> futureUrl = imageService.saveImageOnCloud(file);
43                 url = futureUrl.get();
44             } catch (Exception e) {
45                 e.printStackTrace();
46             }
47             Alessandro Leo, 6 days ago * User story 2 completa
48         }
49
50         article.setIsAccepted(isAccepted:null); You, last week * Uncommitted changes
51
52         ArticleDto dto = modelMapper.map(articleRepository.save(article), destinationType:ArticleDto.class);
53
54         if(!file.isEmpty()){
55             imageService.saveImageOnDB(url, article);
56         }
57
58         return dto;
59     }
60
61 }

```

Da adesso in poi qualsiasi articolo appena creato avrà il flag a null, possiamo fare un prova e vedremo che nella tabella “articles” il nuovo articolo avrà il flag “is\_accepted” a null.

Fatti questi passaggi andiamo ad eliminare quindi la visualizzazione degli articoli non ancora accettati sia dalla home che dall’index di tutti gli articoli.

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java”

```

UserController.java 1 M
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > UserController > articleRepository
You, 1 minute ago | 2 authors (Alessandro Leo and one other)
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.util.ArrayList;
4 import java.util.Collections;
5 import java.util.Comparator;
6 import java.util.List;
7 import java.util.stream.Collectors;
8
9 import org.modelmapper.ModelMapper;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.stereotype.Controller;
12 import org.springframework.validation.BindingResult;
13 import org.springframework.web.bind.annotation.GetMapping;
14 import org.springframework.web.bind.annotation.ModelAttribute;
15 import org.springframework.web.bind.annotation.PathVariable;
16 import org.springframework.web.bind.annotation.PostMapping;
17 import org.springframework.web.bind.annotation.RequestMapping;
18 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
19
20 import it.aulab.progetto_finale_demo_doc.services.ArticleService;
21 import it.aulab.progetto_finale_demo_doc.services.CategoryService;
22 import it.aulab.progetto_finale_demo_doc.services.UserService;
23 import jakarta.servlet.http.HttpServletRequest;
24 import jakarta.servlet.http.HttpServletResponse;
25 import jakarta.validation.Valid;
26 import it.aulab.progetto_finale_demo_doc.models.Article;
27 import it.aulab.progetto_finale_demo_doc.models.User;
28 import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
29 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
30 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
31 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
32
33 @Controller
34 public class UserController {
35
36     @Autowired
37     private UserService userService;
38
39     @Autowired
40     private ArticleService articleService;
41
42     @Autowired
43     private ArticleRepository articleRepository; // I minute ago + uncommitted changes
44
45     @Autowired
46     private CareerRequestRepository careerRequestRepository;
47
48     @Autowired
49     private CategoryService categoryService;
50
51     @Autowired
52     private ModelMapper modelMapper;
53
54     //Rotta di home
55     @GetMapping("/")
56     public String home(Model viewModel) {
57
58         //Recupero tutti gli articoli accettati
59         List<ArticleDto> articles = new ArrayList<ArticleDto>();
60         for(Article article : articleRepository.findByIsAcceptedTrue()){
61             articles.add(modelMapper.map(article, destinationType:ArticleDto.class));
62         }
63
64         //ordino e invio al template gli articoli ordinati in modo decrescente
65         Collections.sort(articles, Comparator.comparing(ArticleDto::getPublishDate).reversed());
66
67         List<ArticleDto> lastThreeArticles = articles.stream().limit(maxSize:3).collect(Collectors.toList());
68
69         viewModel.addAttribute(attributeName:"articles", lastThreeArticles);
70
71         return "home";
72     }
73
74 }

```

Dove abbiamo aggiunto un modo per recuperare solo gli articoli che hanno "isAccepted" a true.

Stiamo utilizzando il model mapper per mappare in dto la risposta di quella che sarà la derivedQuery "findByIsAcceptedTrue" che non abbiamo ancora all'interno del repository.

- Ne approfitto per farvi notare che in questo caso stiamo utilizzando un anti-pattern dove il controller richiede direttamente un metodo al repository senza passare per il service.

Andiamo allora a creare le derived query che ci servono

in "src\main\java\it\aulab\progetto\_finale\_docente\repositories\ArticleRepository.java"

```

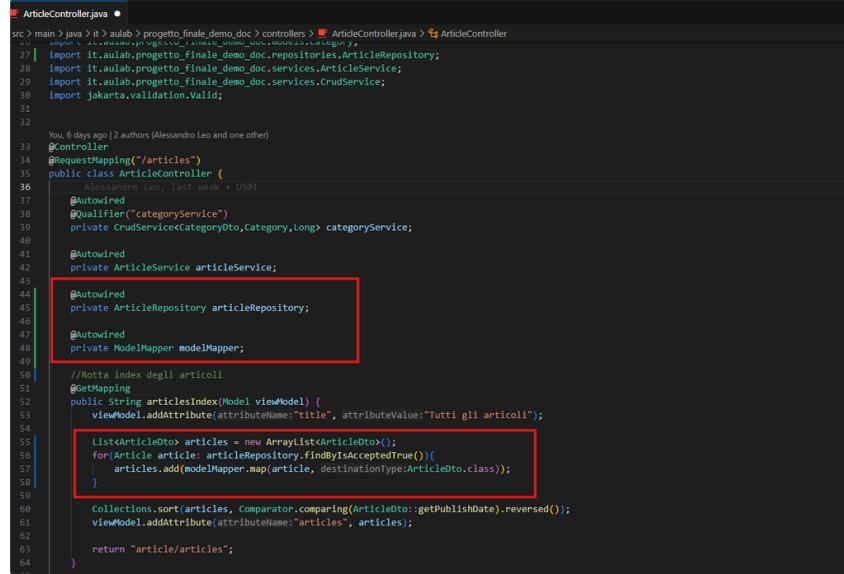
ArticleRepository.java M
src > main > java > it > aulab > progetto_finale_demo_doc > repositories > ArticleRepository.java > ArticleRepository > findByUser(User)
You, 1 second ago | 2 authors (Alessandro Leo and one other)
1 package it.aulab.progetto_finale_demo_doc.repositories;
2
3 import java.util.List;
4
5 import org.springframework.data.repository.ListCrudRepository;
6
7 import it.aulab.progetto_finale_demo_doc.models.Article;
8 import it.aulab.progetto_finale_demo_doc.models.Category;
9 import it.aulab.progetto_finale_demo_doc.models.User;
10
11 public interface ArticleRepository extends ListCrudRepository<Article, Long>{
12     List<Article> findByCategory(Category category);
13     List<Article> findByUser(User user); Alessandro Leo, last week * US01
14     List<Article> findByIsAcceptedTrue();
15     List<Article> findByIsAcceptedFalse();
16     List<Article> findByIsAcceptedIsNull();
17 }

```

Dove abbiamo creato le derived query che coprono i tre casi.

L'handler modificato in "UserController" mostra gli articoli nella home, andiamo a modificare l'handler che mostra gli articoli nella index

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"



```

23 | import org.springframework.web.bind.annotation.*;
24 |
25 | import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
26 | import it.aulab.progetto_finale_demo_doc.services.ArticleService;
27 | import it.aulab.progetto_finale_demo_doc.services.CrudService;
28 | import jakarta.validation.Valid;
29 |
30 |
31 |
32 | You 6 days ago | 2 authors (Alessandro Leo and one other)
33 | @Controller
34 | @RequestMapping("/articles")
35 | public class ArticleController {
36 |     Alessandro Leo, last week + US01
37 |     @Autowired
38 |     @Qualifier("categoryService")
39 |     private CrudService<CategoryDto,Category,Long> categoryService;
40 |
41 |     @Autowired
42 |     private ArticleService articleService;
43 |
44 |     @Autowired
45 |     private ArticleRepository articleRepository;
46 |
47 |     @Autowired
48 |     private ModelMapper modelMapper;
49 |
50 |     //Rotta index degli articoli
51 |     @GetMapping
52 |     public String articlesIndex(Model viewModel) {
53 |         viewModel.addAttribute(attributeName:"title", attributeValue:"Tutti gli articoli");
54 |
55 |         List<ArticleDto> articles = new ArrayList<ArticleDto>();
56 |         for(Article article: articleRepository.findByIsAcceptedTrue()){
57 |             articles.add(modelMapper.map(article, destinationType:ArticleDto.class));
58 |         }
59 |
60 |         Collections.sort(articles, Comparator.comparing(ArticleDto::getPublishDate).reversed());
61 |         viewModel.addAttribute(attributeName:"articles", articles);
62 |
63 |         return "article/articles";
64 |     }

```

Qui stessa cosa del controller precedente dove andiamo sempre ad utilizzare model mapper ed antipattern.

Facciamo un test e vediamo che gli articoli in home non ci sono più, poichè attualmente ancora non accettati e stessa cosa avviene nella index.

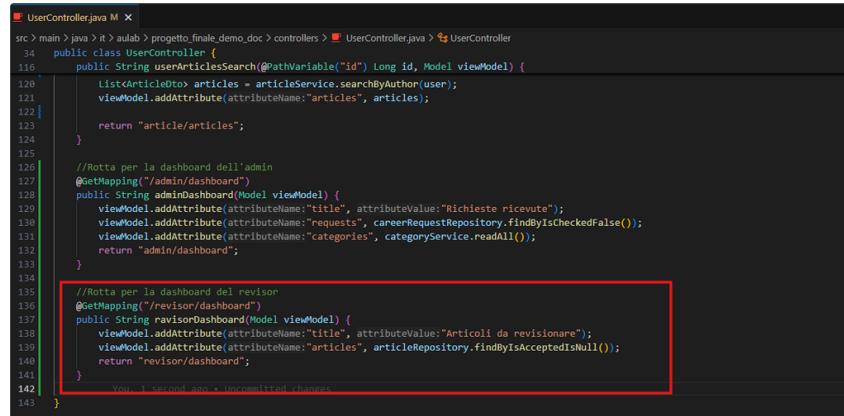
## REVISOR DASHBOARD

Ma perchè abbiamo fatto tutti questi passaggi?

Perchè adesso andremo a costruire una dashboard per il revisore che visualizzerà tutti gli articoli da revisionare e potrà accettarli o rifiutarli.

Iniziamo col creare l'handler all'interno del controller “UserController”

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java”



```

23 | main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > UserController
24 |
25 | public class UserController {
26 |     public String userArticlesSearch(@PathVariable("id") Long id, Model viewModel) {
27 |         List<ArticleDto> articles = articleService.searchByAuthor(user);
28 |         viewModel.addAttribute(attributeName:"articles", articles);
29 |
30 |         return "article/articles";
31 |
32 |
33 |     //Rotta per la dashboard dell'admin
34 |     @GetMapping("/admin/dashboard")
35 |     public String adminDashboard(Model viewModel) {
36 |         viewModel.addAttribute(attributeName:"title", attributeValue:"Richieste ricevute");
37 |         viewModel.addAttribute(attributeName:"requests", careerRequestRepository.findByIsCheckedFalse());
38 |         viewModel.addAttribute(attributeName:"categories", categoryService.readAll());
39 |
40 |         return "admin/dashboard";
41 |     }
42 |
43 |     //Rotta per la dashboard del revisor
44 |     @GetMapping("/revisor/dashboard")
45 |     public String revisionDashboard(Model viewModel) {
46 |         viewModel.addAttribute(attributeName:"title", attributeValue:"Articoli da revisionare");
47 |         viewModel.addAttribute(attributeName:"articles", articleRepository.findByIsAcceptedIsNull());
48 |
49 |         return "revisor/dashboard";
50 |     }
51 |
52 |     ...
53 | }

```

Dove vediamo vengono caricati dal database tutti gli articoli con il campo “is\_accepted” a null.

## TEMPLATE

Costruiamo ora il template in “src\main\resources\templates” dove creeremo un folder “revisor” ed il file che chiameremo “dashboard.html”

in “src\main\resources\templates\revisor\dashboard.html”

```

src > main > resources > templates > revisor > dashboard.html U
1  <!DOCTYPE html>
2  <html xmlns:th="http://www.thymeleaf.org">
3  <head th:insert="~{index :: head}"></head>
4  <body>
5      <head th:insert="~{index :: navbar}"></head>
6
7      <div class="vh-100">
8          <div th:if="${resultMessage}" class="alert alert-success">
9              <p>${th:text=${resultMessage}}</p>
10         </div>
11
12         <div class="container">
13             <div class="row">
14                 <div class="col-12 my-5 d-flex justify-content-center align-items-center">
15                     <span class="h1">Dashboard Revisor</span>
16                 </div>
17
18                 <div class="col-12 my-5 d-flex justify-content-center align-items-center">
19                     <span class="h1">Articoli da revisionare</span>
20                 </div>
21
22                 <div class="col-12">
23                     <table class="table table-striped table-responsive-lg">
24                         <thead>
25                             <tr>
26                                 <th>Id</th>
27                                 <th>Titolo</th>
28                                 <th>SubTitle</th>
29                                 <th>Category</th>
30                                 <th>Actions</th>
31                             </tr>
32                         </thead>
33                         <tbody>
34                             <tr th:each="article: ${articles}">
35                                 <td th:text="${article.id}"></td>
36                                 <td th:text="${article.title}"></td>
37                                 <td th:text="${article.subtitle}"></td>
38                                 <td th:text="${article.category}">${th:text=${article.category.name}}</td>
39                                 <td th:unless="${article.category}">Nessuna categoria</td>
40
41                                 <td>
42                                     <a href="@{/articles/revisor/detail/{id}(id=${article.id})}" class="btn btn-primary m-2">
43                                         leggi
44                                     </a>
45                                 </td>
46                             </tr>
47                         </tbody>
48                     </table>
49                 </div>
50             </div>
51         </div>
52
53         <head th:insert="~{index :: footer}"></head>
54         <script th:replace="~{index :: bootstrapScript}"></script>
55     </body>
56 </html>

```

Questo template mostrerà una tabella all'interno della quale verranno elencati tutti gli articoli da revisionare con la possibilità di leggerli nello specifico. Quindi il tasto leggi aprirà un dettaglio dell'articolo all'interno del quale potremo decidere se accettare o rifiutare l'articolo stesso.

Testiamo il nostro uri tramite l'url "<http://localhost:8080/revisor/dashboard>" e vedremo ciò che abbiamo appena costruito.

**NB:** Non preoccupiamoci dell'accesso a questa dashboard senza che l'utente abbia effettivamente il ruolo di revisore, l'accesso verrà bloccato più avanti

Ma il tasto leggi in realtà fa riferimento ad un handler che ancora non abbiamo creato, procediamo.

Lo creeremo all'interno del controller degli articoli essendo appunto un dettaglio specifico di un articolo

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"

```

ArticleController.java M
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ArticleController > revisorDetailArticle(Long, Model)
35  public class ArticleController {
77      public String articleStore(@Valid @ModelAttribute("article") Article article,
96
97
98      //Rotta di dettaglio di un articolo
99      @GetMapping("detali/{id}")
100     public String detailArticle(@PathVariable("id") Long id, Model viewModel) {
101         viewModel.addAttribute(attributeName:"title", attributeValue:"Article detail");
102         viewModel.addAttribute(attributeName:"article", articleService.read(id));
103         return "article/detail";
104     }
105
106     //Rotta dettaglio di un articolo per il revisore
107     @GetMapping("revisor/detail/{id}")
108     public String revisorDetailArticle(@PathVariable("id") Long id, Model viewModel) {
109         viewModel.addAttribute(attributeName:"title", attributeValue:"Article detail");
110         viewModel.addAttribute(attributeName:"article", articleService.read(id));
111         return "revisor/detail";
112     }
113 }

```

Costruiamo di conseguenza il template di dettaglio per il revisore in "src\main\resources\templates" dove creeremo un file chiamato appunto "detail.html"

in "src\main\resources\templates\revisor\detail.html"

```

  detail.html M X
src > main > resources > templates > revisor > detail.html > HTML > body > div.container-fluid p.sbg-secondary-subtle-test-center > div.col-12
  You 8 minutes ago | 2 authors (Alessandro Leo and one other)
  1 <!DOCTYPE html>
  2 <html xmlns="http://www.w3.org/1999/xhtml">
  3 <head>
  4 <body>
  5   <head> th:insert="{[index :: navbar]}></head>
  6
  7   <div class="container-fluid w-100 bg-secondary-subtle text-center">
  8     <div class="row justify-content-center">
  9       <div class="col-12">
 10         <span class="text-center ArticleTitle">
 11           <span th:text="${article.title}">${titolo}</span>
 12           <div> Alexander Leo, 2 days ago + (0) complete
 13         </div>
 14       </div>
 15     </div>
 16     <div class="container my-5">
 17       <div class="row justify-content-center">
 18         <div class="col-12 col-md-6 flex flex-column">
 19           <div id="carouselExample" class="carousel slide">
 20             <div class="carousel-inner">
 21               <div class="carousel-item active">
 22                 
 23                 
 24               </div>
 25             </div>
 26             <button class="carousel-control-prev" type="button" data-bs-target="#carouselExample" data-bs-slide="prev">
 27               <span class="carousel-control-prev-icon" aria-hidden="true"></span>
 28               <span class="visually-hidden">Previous
 29             </button>
 30             <button class="carousel-control-next" type="button" data-bs-target="#carouselExample" data-bs-slide="next">
 31               <span class="carousel-control-next-icon" aria-hidden="true"></span>
 32               <span class="visually-hidden">Next
 33             </button>
 34           </div>
 35         <div class="text-center">
 36           <h2 th:text="${article.subtitle}">Sottotitolo</h2>
 37
 38           <p class="fs-5" th:if="${article.category != null}">Categoria:<br/>
 39             <a href="#"@{categories/search[id]}(id=${article.category.id})>${textCapitalize ${article.category.name}}</a>
 40           </p>
 41           <p class="fs-5" th:if="${article.category == null}">Nessuna categoria</p>
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79   <head> th:insert="{[index :: footer]}></head>
 80   <script th:replace="{[index :: bootstrapScript]}></script>
 81   <script>
 82     function goBack() {
 83       window.history.back();
 84     }
 85   </script>
 86 </body>
 87 </html>

```

Fatto questo facciamo un test recandoci sempre su "<http://localhost:8080/revisor/dashboard>" e cliccando sul tasto leggi. Ci verrà aperta una nuova pagina che contiene l'articolo con in più i button che ci consentiranno di accettare o rifiutare un articolo, ma che attualmente sono agganciati a degli handler non ancora creati.

## HANLDERS ACCETTA E RIFIUTA

Andiamo quindi ora a gestire il meccanismo di accettazione o rifiuto dell'articolo.

Come vediamo i due button costruiscono un uri dinamicamente tramite query string dal quale possiamo catturare i dati che ci servono ed utilizzarli nel nostro handler.

Andiamo allora a creare l'handler all'interno di "ArticleController"

in "src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java"

```

1 ArticleController.java 2, M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ArticleController > articleSetAccepted(String, Long, RedirectAttributes)
You, 40 seconds ago | 2 authors (Alessandro Leo and one other)
1 package it.aulab.progetto_finale_demo_doc.controllers;
2
3 import java.security.Principal;
4 import java.util.ArrayList;
5 import java.util.Collections;
6 import java.util.Comparator;
7 import java.util.List;
8
9 import org.modelmapper.ModelMapper;
10 import org.springframework.beans.factory.annotation.Autowired;
11 import org.springframework.beans.factory.annotation.Qualifier;
12 import org.springframework.stereotype.Controller;
13 import org.springframework.ui.Model;
14 import org.springframework.validation.BindingResult;
15 import org.springframework.web.bind.annotation.GetMapping;
16 import org.springframework.web.bind.annotation.ModelAttribute;
17 import org.springframework.web.bind.annotation.PathVariable;
18 import org.springframework.web.bind.annotation.PostMapping;
19 import org.springframework.web.bind.annotation.RequestParam;
20 import org.springframework.web.bind.annotation.RequestPart;
21 import org.springframework.web.multipart.MultipartFile;
22 import org.springframework.web.servlet.mvc.support.RedirectAttributes;
23
24 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
25 import it.aulab.progetto_finale_demo_doc.dtos.CategoryDto;
26 import it.aulab.progetto_finale_demo_doc.models.Article;
27 import it.aulab.progetto_finale_demo_doc.models.Category;
28 import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
29 import it.aulab.progetto_finale_demo_doc.services.ArticleService;
30 import it.aulab.progetto_finale_demo_doc.services.CrudService;
31 import jakarta.validation.Valid;
32
33
//Rotta dettaglio di un articolo per il revisore
@GetMapping("revisor/detail/{id}")
public String revisorDetailArticle(@PathVariable("id") Long id, Model viewModel) {
    viewModel.addAttribute(attributeName:"title", attributeValue:"Articolo detail");
    viewModel.addAttribute(attributeName: "article", articleService.read(id));
    return "revisor/detail";
}

//Rotta dedicata all'azione del revisore
@PostMapping("accept")
public String articleSetAccepted(@RequestParam("action") String action, @RequestParam("articleId") long articleId, RedirectAttributes redirectAttributes) {
    if(action.equals("accept")){
        articleService.setIsAccepted(true, articleId);
        redirectAttributes.addFlashAttribute(attributeName:"resultMessage", attributeValue:"Articolo accettato!");
    }else if(action.equals("reject")){
        articleService.setIsAccepted(false, articleId);
        redirectAttributes.addFlashAttribute(attributeName:"resultMessage", attributeValue:"Articolo rifiutato!");
    }else{
        redirectAttributes.addFlashAttribute(attributeName:"resultMessage", attributeValue:"Azione non corretta!");
    }
    return "redirect:/revisor/dashboard";
}

```

Questo metodo valorizzerà il messaggio flash in base alla scelta fatta , ma richiama anche il metodo “setIsAccepted” dell’ “articleService” che non abbiamo creato, quindi andiamo ad implementarlo

in “src\main\java\it\aulab\progetto\_finale\_docente\services\ArticleService.java”

```

1 ArticleService.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > searchByCategory(Category)
26 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
101
102     public List<ArticleDto> searchByCategory(Category category){
103         List<ArticleDto> dtos = new ArrayList<ArticleDto>();
104         for(Article article: articleRepository.findByCategory(category)){
105             dtos.add(modelMapper.map(article, destinationType:ArticleDto.class)); Alessandro Leo, 6 days ago + User
106         }
107         return dtos;
108     }
109
110     public List<ArticleDto> searchByAuthor(User user){
111         List<ArticleDto> dtos = new ArrayList<ArticleDto>();
112         for(Article article: articleRepository.findByName(user)){
113             dtos.add(modelMapper.map(article, destinationType:ArticleDto.class));
114         }
115         return dtos;
116     }
117
118     public void setIsAccepted(Boolean result, long id){
119         Article article = articleRepository.findById(id).get();
120         article.setIsAccepted(result);
121         articleRepository.save(article);
122     }
123
124 }

```

Possiamo fare un test e vedremo che il tutto funziona. Gli articoli una volta accettati verranno visualizzati sia nella home che nella nostra index.

## MODIFICA NAVBAR

Tutto il meccanismo adesso funziona, andiamo quindi ad abilitare il link sulla navbar che sarà però visibile solo ad utenti che hanno ruolo di revisore.

in “src\main\resources\templates\index.html”

```

1 <!-- index.html M x
2 <html xmlns="http://www.thymeleaf.org">
3   <head>
4     <meta charset="UTF-8"/>
5     <title>Aulab Chronicle
6   </head>
7   <body>
8     <nav class="navbar navbar-expand-lg navbar-light bg-light">
9       <div class="container-fluid">
10         <a href="#" class="navbar-brand" th:href="@{'/'}">Aulab Chronicle
11         <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
12           <span class="navbar-toggler-icon">
13         </button>
14         <div class="collapse navbar-collapse" id="navbarSupportedContent">
15           <ul class="navbar-nav me-auto mb-2 mb-lg-0">
16             <li sec:authorize="isAuthenticated" class="nav-item">
17               <a class="nav-link" href="#">Crea articolo</a>
18             </li>
19             <li class="nav-item">
20               <a class="nav-link" href="#">/articles/create">Tutti gli articoli</a>
21             </li>
22             <li class="nav-item">
23               <a class="nav-link" href="#">/articles">sec:authorize="hasRole('ROLE_ADMIN')">
24                 <div class="d-flex">
25                   <a class="nav-link" href="#">/admin/dashboard">Dashboard admin</a>
26                   <div class="mt-1" th:if="${careerRequests > 0}">
27                     <ul class="fas fa-bell fa-1x pt-2" style="list-style-type: none;">
28                       <li></span>
29                     </ul>
30                   </div>
31                 </div>
32               </a>
33             </li>
34             <li class="nav-item" sec:authorize="hasRole('ROLE_REVISOR')">
35               <a class="nav-link" href="#">/revisor/dashboard">Dashboard revisor</a>
36               <div class="mt-1" th:if="${articlesToBeRevised > 0}">
37                 <ul class="fas fa-bell fa-1x pt-2" style="list-style-type: none;">
38                   <li></span>
39                 </ul>
40               </div>
41             </li>
42             <li class="nav-item" sec:authorize="isAuthenticated">
43               <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
44                 Accesso
45               </a>
46               <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
47                 <li sec:authorize="!anonymous" class="dropdown-item" aria-current="page" th:href="@{/register}">Register</a>
48                 <li sec:authorize="!anonymous" class="dropdown-item" aria-current="page" th:href="@{/login}">Login</a>
49                 <li sec:authorize="isAuthenticated" class="dropdown-item" aria-current="page" th:href="@{/logout}">Logout</a>
50               </ul>
51             </li>
52           </ul>
53         </div>
54       </div>
55     </nav>
56   </body>
57 </html>

```

Anche per questo link abbiamo inserito il sistema di notifica tramite campanellina ma per renderlo funzionante ci basterà aggiungere un nuovo dato da condividere all'interno di "NotificationInterceptor.java"

in "src\main\java\it\aulab\progetto\_finale\_docente\config\NotificationInterceptor.java"

```

1 <!-- NotificationInterceptor.java U x
2 <!-- main > java > it > aulab > progetto_finale_demo_doc > config > NotificationInterceptor.java > NotificationInterceptor > articleRepository
3 package it.aulab.progetto_finale_demo_doc.config;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Component;
7 import org.springframework.web.servlet.HandlerInterceptor;
8 import org.springframework.web.servlet.ModelAndView;
9
10 import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
11 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
12 import jakarta.servlet.http.HttpServletRequest;
13 import jakarta.servlet.http.HttpServletResponse;
14
15 @Component
16 public class NotificationInterceptor implements HandlerInterceptor {
17
18   @Autowired
19   CareerRequestRepository careerRequestRepository;
20
21   @Autowired
22   ArticleRepository articleRepository;
23
24   @Override
25   public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
26                         ModelAndView modelAndView) throws Exception {
27     if (modelAndView != null && request.isUserInRole(role: "ROLE_ADMIN")) {
28       int careerCount = careerRequestRepository.findByIsCheckedFalse().size();
29       modelAndView.addObject(attributeName: "careerRequests", careerCount);
30
31       if (modelAndView != null && request.isUserInRole(role: "ROLE_REVISOR")) {
32         int revisedCount = articleRepository.findByIsAcceptedIsNull().size();
33         modelAndView.addObject(attributeName: "articlesToBeRevised", revisedCount);
34       }
35     }
36   }
37 }

```

Fatto questo possiamo fare un test registrandoci con un nuovo utente e abilitando il ruolo di revisore. Vedremo le notifiche e potremo accedere alla nostra dashboard da revisore.

**⚠ IMPORTANTE:** Nota sui ruoli. Se si assegna un ruolo manualmente tramite DB ad un utente, o si effettua qualsiasi operazione su un utente per cambiargli il ruolo, se l'utente è loggato non basterà ricaricare la pagina per assegnare i nuovi permessi, ma OGNI VOLTA che un utente riceve un cambio ruolo (o una aggiunta) bisogna sloggarsi e riloggarsi così che vengano assegnate le authorities.

## BLOCCARE LE ROTTE

Abbiamo però un problema di sicurezza, come già affrontato nei test precedenti. Le rotte dedicate al revisore devono essere accessibili solo ad un utente che ha tale ruolo.

Andiamo quindi nella nostra configurazione di sicurezza e blocchiamo le rotte.

in "src\main\java\it\aulab\progetto\_finale\_docente\config\SecurityConfig.java" modifichiamo

```

  1 SecurityConfig.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > config > SecurityConfig.java > SecurityConfig > filterChain(HttpServletRequest)
  20 public class SecurityConfig {
  21
  22     @Bean
  23     public SecurityFilterChain filterChain(HttpServletRequest http) throws Exception {
  24
  25         http
  26             .csrf(csrf -> csrf.disable())
  27             .authorizeHttpRequests(authorize ->
  28                 authorize.requestMatchers(.::patterns:"/register/**").permitAll()
  29                 .requestMatchers(.::patterns:"/admin/dashboard","/categories/create","/categories/edit/{id}","/categories/update/{id}","/categories/delete/{id}").hasRole(role:"ADMIN")
  30                 .requestMatchers(.::patterns:"/revisor/dashboard","/revision/detail/{id}","/accept").hasRole(role:"REVISOR")
  31                 .anyRequest().authenticated()
  32             ).formLogin(form ->
  33                 form.loginPage("/login")
  34                 .loginProcessingUrl("/login")
  35                 .defaultSuccessUrl("/")
  36                 .defaultSuccessUrl("defaultSuccessUrl"))
  37             ).permitAll()
  38         ).logout(logout ->
  39             logout.logoutSuccessHandler(new AntPathRequestMatcher(pattern:"/logout"))
  40             .permitAll()
  41             .exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl:"/error/403"))
  42             .sessionManagement(session -> session
  43                 .maximumSessions(maximumSessions:1)
  44                 .maxSessionDuration(maxSessionDuration:10000)
  45                 .expiredUrl(expiredUrl:"/login?session-expired=true")
  46             );
  47         );
  48     }
  49
  50     return http.build();
  51 }
  52

```

Con questo ultimo passaggio abbiamo protetto le rotte affinché solo un revisore possa averci accesso.

## VISUALIZZAZIONE NELLA RICERCA DI SOLO ARTICOLI ACCETTATI

Esiste un ulteriore problema però. Attualmente quando ricerchiamo per autore o categoria , nelle visualizzazioni di dettagli, ci verranno visualizzati anche gli articoli non accettati.

Modifichiamo questo comportamento.

Il primo da modifcare è nel “CategoryController”

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\ArticleController.java”

```

  1 CategoryController.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > CategoryController.java > CategoryController > categorySearch(Long, Model)
  27 public class CategoryController {
  28     @Autowired
  29     private CategoryService categoryService;
  30
  31     @Autowired
  32     private ModelMapper modelMapper;
  33
  34     //Rotta per la ricerca dell'articolo in base alla categoria
  35     @GetMapping("search/{id}")
  36     public String categorySearch(@PathVariable("id") long id, Model viewModel) {
  37         CategoryDto category = categoryService.read(id);
  38
  39         viewModel.addAttribute(attributeName:"title", "Tutti gli articoli trovati per categoria " + category.getNome());
  40
  41         List<ArticleDto> articles = articleService.searchByCategory(modelMapper.map(category, destinationType:Category.class));
  42
  43         List<ArticleDto> acceptedArticles = articles.stream().filter(article -> Boolean.TRUE.equals(article.getIsAccepted())).collect(Collectors.toList());
  44
  45         viewModel.addAttribute(attributeName:"articles", acceptedArticles);
  46
  47         return "article/articles";
  48     }
  49
  50 }
  51
  52

```

Il secondo è nello “UserController”

in “src\main\java\it\aulab\progetto\_finale\_docente\controllers\UserController.java”

```

  1 UserController.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > UserController > userArticlesSearch(Long, Model)
  35 public class UserController {
  36     public String registration(@Valid @ModelAttribute("user") UserDto userDto,
  37
  38         ) {
  39             return "redirect:/";
  40         }
  41
  42         //Rotta per la ricerca degli articoli in base all'utente
  43         @GetMapping("search/{id}")
  44         public String userArticlesSearch(@PathVariable("id") long id, Model viewModel) {
  45             User user = userService.findById();
  46             viewModel.addAttribute(attributeName:"title", "Tutti gli articoli trovati per utente " + user.getUsername());
  47
  48             List<ArticleDto> articles = articleService.searchByAuthor(user);
  49
  50             List<ArticleDto> acceptedArticles = articles.stream().filter(article -> Boolean.TRUE.equals(article.getIsAccepted())).collect(Collectors.toList());
  51
  52             viewModel.addAttribute(attributeName:"articles", acceptedArticles);
  53
  54             return "article/articles";
  55         }
  56
  57

```