

Java - User Story 5 - PDF

- Come Sara
- vorrei poter cancellare o modificare gli articoli che ho scritto
- in modo tale da consegnare sempre un lavoro d'alta qualità

ACCEPTANCE CRITERIA:

- Creazione di una dashboard dedicata ai writer
- Permettere solo al writer specifico la modifica di un articolo
- Permettere solo al writer specifico la cancellazione di un articolo

EXTRA:

- Se l'articolo viene modificato, deve ritornare alla revisione

Svolgimento

In questa user story procederemo con la creazione del meccanismo riservato solo al writer per la scrittura dell'articolo la modifica e la cancellazione dello stesso.

Iniziamo allora con la creazione della dashboard a lui dedicata.

Per prima cosa partiamo dal controller "UserController"

in "src\main\java\it\aulab\progetto_finale_docente\controllers\UserController.java"

```
UserController.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > UserController.java > ...
You, 20 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 import java.util.stream.Collectors;
9
10 import org.modelmapper.ModelMapper;
11 import org.springframework.beans.factory.annotation.Autowired;
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.servlet.mvc.support.RedirectAttributes;
20
21 import it.aulab.progetto_finale_demo_doc.services.ArticleService;
22 import it.aulab.progetto_finale_demo_doc.services.CategoryService;
23 import it.aulab.progetto_finale_demo_doc.services.UserService;
24 import jakarta.servlet.http.HttpServletRequest;
25 import jakarta.servlet.http.HttpServletResponse;
26 import jakarta.validation.Valid;
27 import it.aulab.progetto_finale_demo_doc.models.Article;
28 import it.aulab.progetto_finale_demo_doc.models.User;
29 import it.aulab.progetto_finale_demo_doc.repositories.ArticleRepository;
30 import it.aulab.progetto_finale_demo_doc.repositories.CareerRequestRepository;
31 import it.aulab.progetto_finale_demo_doc.dtos.ArticleDto;
32 import it.aulab.progetto_finale_demo_doc.dtos.UserDto;
33
34 @Controller
35 public class UserController {
36
```

```

135 //Rotta per la dashboard del revisor
136 @GetMapping("/revisor/dashboard")
137 public String revisorDashboard(Model viewModel) {
138     viewModel.addAttribute(attributeName:"title", attributeValue:"Articoli da revisionare");
139     viewModel.addAttribute(attributeName:"articles", articleRepository.findByIsAcceptedIsNull());
140     return "revisor/dashboard";
141 }
142
143 //Rotta per la dashboard del writer
144 @GetMapping("/writer/dashboard")
145 public String writerDashboard(Model viewModel , Principal principal) {
146
147     viewModel.addAttribute(attributeName:"title", attributeValue:"I tuoi articoli");
148
149     List<ArticleDto> userArticles = articleService.readAll()
150                                     .stream()
151                                     .filter(article -> article.getUser().getEmail().equals(principal.getName()))
152                                     .toList();
153
154     viewModel.addAttribute(attributeName:"articles", userArticles);
155
156     return "writer/dashboard";
157 }
158
159 }
160

```

Questo metodo andrà a filtrare gli articoli della tabella articles in tal modo che vengano estratti solo quelli appartenenti, e quindi scritti, dall'utente loggato che visualizza la dashboard.

L'handler richiamerà il template dashboard all'interno del folder "writer" che attualmente non esiste.

Andiamo quindi in "src/main/resources/templates" dove creiamo il nuovo folder "writer" con all'interno il file "dashboard.html"

in "src/main/resources/templates/writer/dashboard.html"

```

src > main > resources > templates > writer > dashboard.html > 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="min-vh-100">
8 <div th:if="${successMessage}" class="alert alert-success">
9 <p th:text="${successMessage}"></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 Writer</span>
16 </div>
17
18 <div class="col-12 my-5 d-flex justify-content-center align-items-center">
19 <span class="h1" th:text="${title}">I tuoi articoli</span>
20 </div>
21
22
23 <div class="col-12">
24 <table class="table table-striped table-responsive-lg">
25 <thead>
26 <tr>
27 <th>Id</th>
28 <th>Title</th>
29 <th>Subtitle</th>
30 <th>Category</th>
31 <th>#Actions</th>
32 </tr>
33 </thead>
34 <tbody>
35 <tr th:each="article:${articles}">
36 <td th:text="${article.id}"></td>
37 <td th:text="${article.title}"></td>
38 <td th:text="${article.subtitle}"></td>
39 <td th:if="${article.category}" th:text="${article.category.name}"></td>
40 <td th:unless="${article.category}">nessuna categoria</td>
41 <td>
42 <a th:href="@{/articles/detail/{id}(id=${article.id})}" class="btn btn-primary m-2">
43 <i class="fa-solid fa-bars"></i>
44 </a>
45 <a th:href="@{/articles/edit/{id}(id=${article.id})}" class="btn btn-warning m-2">
46 <i class="fa-solid fa-pen"></i>
47 </a>
48
49 <a th:href="@{/articles/delete/{id}(id=${article.id})}" class="btn btn-danger m-2">
50 <i class="fa-solid fa-trash"></i>
51 </a>
52 </td>
53 </tr>
54 </tbody>
55 </table>
56 </div>
57 </div>
58 </div>
59
60 <head th:insert="~{index :: footer}"></head>
61 <script th:replace="~{index :: bootstrapScript}"></script>
62
63 </body>
64 </html>

```

In questo template verranno elencati in una tabella tutti gli articoli creati dal writer, in più abbiamo già predisposto tutti gli uri collegati ai tasti d'azione per la lettura tramite dettaglio, la modifica e la cancellazione dell'articolo stesso.

Facciamo un test verso il nuovo handler **con un utente registrato** verso l'url "<http://localhost:8080/writer/dashboard>" e ci berrà mostrata la dashboard appena creata. Possiamo anche provare la rotta di dettaglio poichè l'unica attualmente

Nessun problema, procediamo con l'implementazione.

in "src\main\java\it\aulab\progetto_finale_docente\controllers\ArticleController.java"

in "src\main\resources\templates\article\edit.html"

```
uc: main > resources > templates > article > edit.html > <html> <body> <div.min-vh-100> <div.container> <div.row> <div.col-12.my-5> <form> <div.mb-3> <input.date.form-control>
```

```
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="min-vh-100">
8 <div class="container-fluid p-5 bg-secondary-subtle text-center">
9 <div class="row justify-content-center">
10 <div class="col-12">
11 <span class="h1">Modifica articolo</span>
12 <h2 th:text="${article.title}">Titolo</h2>
13 </div>
14 </div>
15 </div>
16
17 <div class="container">
18 <div class="row">
19 <div class="col-12 my-5">
20 <form th:action="@{/articles/update/{id}(id=${article.id})}" method="POST" th:object="${article}" enctype="multipart/form-data">
21 <div class="row mb-3">
22 <div class="col">
23 <label for="articleTitle" class="form-label">Titolo</label>
24 <input id="articleTitle" type="text" class="form-control" th:field="*{title}"
25 placeholder="Inserisci un articolo..."
26 <? th:errors = "*" {title} class="text-danger" th:if="${#fields.hasErrors('title')}"></p>
27 </div>
28 <div class="col">
29 <label for="subtitle" class="form-label">Sottotitolo</label>
30 <input id="subtitle" type="text" class="form-control" th:field="*{subtitle}"
31 placeholder="Inserisci un sottotitolo..."
32 <? th:errors = "*" {subtitle} class="text-danger" th:if="${#fields.hasErrors('subtitle')}"></p>
33 </div>
34 </div>
35 <div class="mb-3">
36 <label for="body" class="form-label">Articolo</label>
37 <input id="body" type="text" class="form-control" th:field="*{body}"
38 placeholder="Inserisci un testo..."
39 <? th:errors = "*" {body} class="text-danger" th:if="${#fields.hasErrors('body')}"></p>
40 </div>
41
42 <div class="mb-3">
43 <label for="date" class="form-label">Publish date (8 chars)</label>
44 <input id="date" type="date" class="form-control" th:field="*{publishDate}"
45 placeholder="yyyyMMdd">
46 </div>
```

```

48         <div class="mb-3">
49             <label for="category" class="form-label">Category:</label>
50             <select id="category" th:field="*{category}" class="form-select">
51                 <option th:each="category:${categories}" th:value="{category.id}"
52                     th:text="{category.name}"></option>
53                 <p th:errors = "{category}" class="text-danger" th:if="{#fields.hasErrors('category')}"></p>
54             </select>
55         </div>
56
57         <div class="mb-3 d-flex flex-column" th:if="{article.image != null}">
58             <label for="image" class="form-label">Immagine dell'articolo:</label>
59             
60         </div>
61         <div th:unless="{article.image != null}">
62             <label for="image" class="form-label">Immagine dell'articolo:</label>
63             <p>Immagine non presente</p>
64         </div>
65
66
67         <div class="mb-3">
68             <label for="image" class="form-label">Image:</label>
69             <input type="file" name="file" id="image" class="form-control"/>
70         </div>
71
72         <button type="submit" class="btn btn-success">Modifica articolo</button>
73     </form>
74 </div>
75 </div>
76 </div>
77 </div>
78 </div>
79
80 <head th:insert="~{index :: footer}"></head>
81 <script th:replace="~{index :: bootstrapScript}"></script>
82 </body>
83 </html>

```

Questo template è un pochino diverso da quello di creazione poichè mostra prima di tutto i campi pre-esistenti ma anche una eventuale immagine pre-esistente.

Possiamo fare un test sulla dashboard del writer e cliccare sul tasto di modifica, vedremo che il form di modifica ci verrà correttamente mostrato.

Abilitiamo ora la modifica dell'articolo.

Il form appena creato richiama un handler nel controller non ancora esistente, procediamo con la creazione in "src\main\java\it\aulab\progetto_finale_docente\controllers\ArticleController.java"

```

ArticleController.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ArticleController > articleUpdate(Long, Article, BindingResult, RedirectAttributes, Principal, MultipartFile, Model)
177
178 public class ArticleController {
179
180     //Rotta di modifica di un articolo
181     @PostMapping("/edit/{id}")
182     public String editArticle(@PathVariable("id") Long id, Model viewModel) {
183         viewModel.addAttribute(attributeName:"title", attributeValue:"Article update");
184         viewModel.addAttribute(attributeName:"article", articleService.read(id));
185         viewModel.addAttribute(attributeName:"categories", categoryService.readAll());
186         return "article/edit";
187     }
188
189     //Rotta di memorizzazione modifica di un articolo
190     @PostMapping("/update/{id}")
191     public String articleUpdate(@PathVariable("id") Long id,
192                               @Valid @ModelAttribute("article") Article article,
193                               BindingResult result,
194                               RedirectAttributes redirectAttributes,
195                               Principal principal,
196                               MultipartFile file,
197                               Model viewModel) {
198
199         //Controllo degli errori con validazioni
200         if (result.hasErrors()) {
201             viewModel.addAttribute(attributeName:"title", attributeValue:"Article update");
202             article.setImage(articleService.read(id).getImage());
203             viewModel.addAttribute(attributeName:"article", article);
204             viewModel.addAttribute(attributeName:"categories", categoryService.readAll());
205             return "article/edit";
206         }
207
208         articleService.update(id, article, file);
209         redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Articolo modificato con successo!");
210
211         return "redirect:/articles";
212     }
213 }

```

Ma l'handler appena creato richiama il metodo "update" dell'ArticleService che ricordiamo non avere ancora la giusta logica.

in "src\main\java\it\aulab\progetto_finale_docente\services\ArticleService.java"

```

ArticleService.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > update(Long, Article, MultipartFile)
26 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
90
91     @Override
92     public ArticleDto update(Long key, Article updatedArticle, MultipartFile file) {
93         String url="";
94
95         //Controllo l'esistenza dell'articolo in base al suo id
96         if (articleRepository.existsById(key)) {
97             //Assegno all'articolo proveniente dal form lo stesso id dell'articolo originale
98             updatedArticle.setId(key);
99             //Recupero l'articolo originale non modificato
100             Article article = articleRepository.findById(key).get();
101             //Imposto l'utente dell'articolo del form con l'utente dell'articolo originale
102             updatedArticle.setUser(article.getUser());
103
104             //Faccio un controllo sulla presenza o meno del file nell'articolo del form quindi capisco se devo modificare o meno l'immagine
105             if(!file.isEmpty()){
106                 try {
107                     //Elimino l'immagine precedente
108                     imageService.deleteImage(article.getImage().getPath());
109                     try {
110                         //Salvo la nuova immagine
111                         CompletableFuture<String> futureUrl = imageService.saveImageOnCloud(file);
112                         url = futureUrl.get();
113                     } catch (Exception e) {
114                         e.printStackTrace();
115                     }
116                     //Salvo il nuovo path nel db
117                     imageService.saveImageOnDB(url, updatedArticle);
118
119                     //Essendo l'immagine modificata l'articolo torna in revisione
120                     updatedArticle.setIsAccepted(isAccepted:null);
121                     return modelMapper.map(articleRepository.save(updatedArticle), destinationType:ArticleDto.class);
122                 } catch (Exception e) {
123                     e.printStackTrace();
124                 }
125             }else if(article.getImage() == null){ //se l'articolo originale non ha un'immagine e nemmeno quello da modificare allora sicuramente non è stata fatta alcuna modifica
126                 updatedArticle.setIsAccepted(article.getIsAccepted());
127             }else{
128                 //Se l'immagine non è stata modificata devo fare un check su tutti gli altri campi se diversi l'articolo torna in revisione
129
130                 //Se l'immagine non è stata modificata posso impostare sull'articolo modificato la stessa immagine dell'articolo di originale
131                 updatedArticle.setImage(article.getImage());
132
133                 if(updatedArticle.equals(article)==false){
134                     updatedArticle.setIsAccepted(isAccepted:null);
135                 }else{
136                     updatedArticle.setIsAccepted(article.getIsAccepted());
137                 }
138             }
139
140             return modelMapper.map(articleRepository.save(updatedArticle), destinationType:ArticleDto.class) ;
141         } else {
142             throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
143         }
144         return null;
145     }

```

Questo metodo di update è più complesso rispetto a quello di creazione, poiché include controlli sulla presenza o meno di una nuova immagine nel form ma anche se l'immagine sull'articolo originale non è mai stata inserita. Questo significa che, se l'utente dovesse decidere di cambiare l'immagine pre-esistente, il metodo si occuperà di eliminare l'immagine precedente dallo storage online. Successivamente, allinea il model da salvare con le modifiche apportate e, in caso di aggiornamento, invia nuovamente l'articolo in revisione.

Per fare questo, è necessario sovrascrivere il metodo equals nel DAO (Data Access Object) dell'entità Article. Utilizzando il metodo equals predefinito di Object, infatti, non otterremmo il risultato desiderato. La sovrascrittura del metodo equals ci permetterà di definire i criteri specifici per considerare due istanze di Article come equivalenti, facilitando così il processo di verifica delle modifiche.

Andiamo quindi in "article" e creiamo il metodo sovrascritto di "equals"

in "src\main\java\it\aulab\progetto_finale_docente\models\Article.java"

```

Article.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > models > Article.java > Article
28 public class Article {
44     @NotEmpty
45     @Size(max = 1000)
46     private String body;
47
48     @Column(nullable = true, length = 8)
49     @NotNull
50     private LocalDate publishDate;
51
52     @Column(nullable = true)
53     private Boolean isAccepted;
54     // Alessandro Leo, 2 weeks ago • US81
55
56     @ManyToOne
57     @JoinColumn(name = "user_id")
58     @JsonIgnoreProperties({"articles"})
59     private User user;
60
61     @ManyToOne
62     @JsonIgnoreProperties({"articles"})
63     private Category category;
64
65     @OneToOne(mappedBy = "article")
66     @JsonIgnoreProperties({"article"})
67     private Image image;
68
69     @Override
70     public boolean equals(Object obj) {
71         Article article = (Article) obj;
72
73         if (title.equals(article.getTitle()) &&
74             subtitle.equals(article.getSubtitle()) &&
75             body.equals(article.getBody()) &&
76             publishDate.equals(article.getPublishDate()) &&
77             category.getName().equals(article.getCategory().getName()) &&
78             image.getPath().equals(article.getImage().getPath())) {
79             return true;
80         }
81
82         return false;
83     }
84
85 }

```

Fatti questi passaggi possiamo procedere con test sulla modifica provando tutte le combinazioni possibili e vedremo che solo se l'articolo è stato modificato tornerà in revisione e le immagini cambiate spariranno dallo storage sostituite da quelle nuove.

DELETE

Procediamo infine con la cancellazione dell'articolo.

Partiamo sempre dal controller "ArticleController" all'interno del quale creiamo l'handler per la delete.

in "src\main\java\it\aulab\progetto_finale_docente\controllers\ArticleController.java"

```

ArticleController.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > controllers > ArticleController.java > ArticleController > articleUpdate(Long, Article, BindingResult, RedirectAttributes, Principal, MultipartFile, Model)
37 public class ArticleController {
116
117     //Rotta di memorizzazione modifica di un articolo
118     @PostMapping("/update/{id}")
119     public String articleUpdate(@PathVariable("id") Long id,
120                               @Valid @ModelAttribute("article") Article article,
121                               BindingResult result,
122                               RedirectAttributes redirectAttributes,
123                               Principal principal,
124                               MultipartFile file,
125                               Model viewModel) {
126
127         //Controllo degli errori con validazioni
128         if (result.hasErrors()) {
129             viewModel.addAttribute(attributeName="title", attributeValue:"Article update");
130             article.setImage(articleService.read(id).getImage());
131             viewModel.addAttribute(attributeName="article", article);
132             viewModel.addAttribute(attributeName="categories", categoryService.readAll());
133             return "article/edit";
134         }
135
136         articleService.update(id, article, file);
137         redirectAttributes.addFlashAttribute(attributeName="successMessage", attributeValue:"Articolo modificato con successo!");
138
139         return "redirect:/articles";
140     }
141
142     //Rotta per la cancellazione di un articolo
143     @GetMapping("/delete/{id}")
144     public String articleDelete(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {
145
146         articleService.delete(id);
147         redirectAttributes.addFlashAttribute(attributeName="successMessage", attributeValue:"Articolo cancellato con successo!");
148
149         return "redirect:/writer/dashboard";
150     }
151 }

```

Questo handler però richiede l'utilizzo del metodo delete di "ArticleService" che attualmente non possiede la giusta logica, procediamo.

in "src/main/java/it/aulab/progetto_finale_docente/services/ArticleService.java"

```
ArticleService.java M X
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > delete(Long)
26 public class ArticleService implements CrudService<ArticleDto, Article, Long>{
91 public ArticleDto update(Long key, Article updatedArticle, MultipartFile file) {
142 }
143 return null;
144 }
145
146 @Override
147 public void delete(Long key) {
148     if (articleRepository.existsById(key)) {
149
150         Article article = articleRepository.findById(key).get();
151
152         try {
153             String path = article.getImage().getPath();
154             article.getImage().setArticle(article:null);
155             imageService.deleteImage(path);
156         } catch (Exception e) {
157             e.printStackTrace();
158         }
159
160         articleRepository.deleteById(key);
161     } else {
162         throw new RuntimeException(HttpStatus.BAD_REQUEST);
163     }
164 }
165 }
```

Con questo ultimo passaggio abbiamo completato tutto il CRUD sull'articolo.

Possiamo effettuare ora un test dove cancellando i nostri articoli anche le immagini verranno automaticamente cancellate dallo storage.

NAVBAR

Creiamo un collegamento sulla navbar verso la dashboard del writer così come abbiamo già fatto per l'admin ed il revisor. Anche questo collegamento faremo in modo che sia visibile solo ad utenti con ruolo writer.

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

```
index.html
src > main > resources > templates > index.html > html > body > nav.navbar.navbar-expand-lg.navbar-light.bg-light > div.container-fluid > div.navbarSupportedContent.collapse.navbar-collapse > ul.navbar-nav.me-auto.mb-2.m
2 <html xmlns:th="http://www.thymeleaf.org">
15 <body>
17 <nav th:fragment="navbar" class="navbar navbar-expand-lg navbar-light bg-light">
18 <div class="container-fluid">
23 <div class="collapse navbar-collapse" id="navbarSupportedContent">
39 </div>
40 <li class="nav-item" sec:authorize="hasRole('ROLE_REVISOR')">
41 <div class="d-flex">
42 <a class="nav-link" href="/revisor/dashboard">Dashboard revisor</a>
43 <div class="mt-1" th:if="${articlesToBeRevised > 0}">
44 <i class="fas fa-bell fa-l pt-2"></i>
45 <span class="badge rounded-pill bg-danger text-white px-1" th:text="${articlesToBeRevised}></span>
46 </div>
47 </div>
48 </li>
49 <li class="nav-item" sec:authorize="hasRole('ROLE_WRITER')">
50 <a class="nav-link" href="/writer/dashboard">Dashboard writer</a>
51 </li>
52 <li class="nav-item dropdown">
53 <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
54 Accesso
55 </a>
56 <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
57 <li sec:authorize="isAnonymous"><a class="dropdown-item" aria-current="page" th:href="@{/register}">Register</a></li>
58 <li sec:authorize="isAnonymous"><a class="dropdown-item" aria-current="page" th:href="@{/login}">Login</a></li>
59 <li sec:authorize="isAuthenticated"><a class="dropdown-item" aria-current="page" th:href="@{/logout}">Logout</a></li>
60 </ul>
61 </li>
62 <div sec:authorize="isAuthenticated" th:text="Benvenuto: ' + ${#authentication.principal.fullname} + ' - ' + ${#authentication.principal.authorities[0].authority.replace
('ROLE_', '')}>Benvenuto:</div>
64 </div>
65 </div>
```

Con questa modifica notiamo immediatamente che il link alla dashboard del writer è disponibile solo a utenti con il ruolo assegnato.

Ma abbiamo ancora qualche miglioria da fare.

Attualmente qualsiasi ruolo può creare un articolo, e questo è sbagliato poichè solo chi ha il ruolo di writer dovrebbe avere il permesso di poter scrivere un articolo.

Procediamo quindi col far sparire il collegamento al form di creazione dell'articolo ad utenti che non sono writer.

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

```

index.html M X
src > main > resources > templates > index.html > html > body > nav.navbar.navbar-expand-lg.navbar-light.bg-light > div.container-fluid > div.navbarSupportedContent.collapse.navbar-collapse > ul.navbar-nav.me-auto.mb-2
2
<html xmlns:th="http://www.thymeleaf.org">
15
<body>
16
17
<nav th:fragment="navbar" class="navbar navbar-expand-lg navbar-light bg-light">
18
<div class="container-fluid">
19
<a class="navbar-brand" th: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">
25
<li sec:authorize="isAuthenticated && hasRole('ROLE_WRITER')" class="nav-item">
26
<a class="nav-link" href="/articles/create">Crea articolo</a>
27
</li>
28
<li class="nav-item">
29
<a class="nav-link" href="/articles">Tutti gli articoli</a>
30
</li>
31
<li class="nav-item" sec:authorize="hasRole('ROLE_ADMIN')">
32
<div class="d-flex">
33
<a class="nav-link" href="/admin/dashboard">Dashboard admin</a>
34
<div class="mt-1" th:if="${careerRequests > 0}">
35
<span class="badge rounded-pill bg-danger text-white px-1" th:text="${careerRequests}"></span>
36
</div>
37
</div>
38
</li>
39
<li class="nav-item" sec:authorize="hasRole('ROLE_REVISOR')">
40
<div class="d-flex">
41
<a class="nav-link" href="/revisor/dashboard">Dashboard revisor</a>
42
<div class="mt-1" th:if="${articlesToBeRevised > 0}">
43
<span class="fas fa-bell fa-1 pt-2"></span>
44
<span class="badge rounded-pill bg-danger text-white px-1" th:text="${articlesToBeRevised}"></span>
45
</div>
46
</div>
47
</li>
48
</ul>

```

SECURITY

Ma abbiamo sempre lo stesso problema di sicurezza. Pur avendo nascosto i collegamenti in realtà gli uri non sono protetti, quindi possiamo accedervi senza avere le autorizzazione.

Andiamo quindi all'interno della nostra configurazione di sicurezza.

in "src\main\java\it\aulab\progetto_finale_docente\config\SecurityConfig.java" e modifichiamo

```

SecurityConfig.java
src > main > java > it > aulab > progetto_finale_demo_doc > config > SecurityConfig.java > SecurityConfig > filterChain(HttpSecurity)
20
public class SecurityConfig {
21
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
}

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(authorize ->
            .requestMatchers(...patterns: "/register/**").permitAll()
            .requestMatchers(...patterns: "/admin/dashboard", "/categories/create", "/categories/edit/{id}", "/categories/update/{id}", "/categories/delete/{id}").hasRole("ADMIN")
            .requestMatchers(...patterns: "/revisor/dashboard", "/articles/create", "/articles/edit/{id}", "/articles/update/{id}", "/articles/delete/{id}").hasRole("WRITER")
            .requestMatchers(...patterns: "/register", "/", "/articles", "/images/**", "/articles/delete/**", "/categories/search/{id}", "/search/{id}", "/articles/search").permitAll()
        )
        .formLogin(form ->
            .loginPage(loginPage: "/login")
            .loginProcessingUrl(loginProcessingUrl: "/login")
            .defaultSuccessUrl(defaultSuccessUrl: "/")
            .permitAll()
        )
        .logout(logout -> logout
            .logoutRequestMatcher(new AntPathRequestMatcher(pattern: "/logout"))
            .permitAll()
        )
        .exceptionHandling(exception -> exception.accessDeniedPage(accessDeniedUrl: "/error/403"))
        .sessionManagement(session -> session
            .sessionCreationPolicy(sessionCreationPolicy: IF_REQUIRED)
            .maximumSessions(maximumSessions: 1)
            .expiredUrl(expiredUrl: "/login?session-expired=true")
        );
    return http.build();
}

```