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

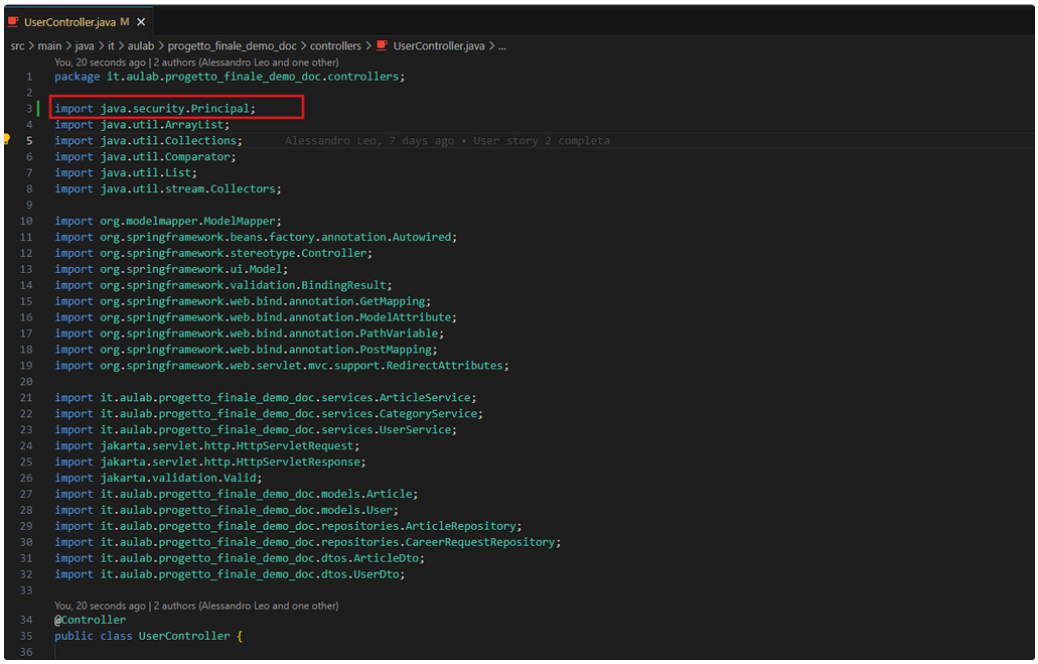
Svolgimento

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"



```java
    //Rotta per la dashboard del revisor
    @GetMapping("/revisor/dashboard")
    public String ravisorDashboard(Model viewModel) {
        viewModel.addAttribute(attributeName:"title", attributeValue:"Articoli da revisionare");
        viewModel.addAttribute(attributeName:"articles", articleRepository.findByIsAcceptedIsNull());
        return "revisor/dashboard";
    }

    //Rotta per la dashboard del writer
    @GetMapping("/writer/dashboard")
    public String writerDashboard(Model viewModel , Principal principal) {

        viewModel.addAttribute(attributeName:"title", attributeValue:"I tuoi articoli");

        List<ArticleDto> userArticles = articleService.readAll()
                                        .stream()
                                        .filter(article -> article.getUser().getEmail().equals(principal.getName()))
                                        .toList();

        viewModel.addAttribute(attributeName:"articles", userArticles);

        return "writer/dashboard";
    }

}
```

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"

```
dashboard.html U
src > main > resources > templates > writer > dashboard.html > html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head th:insert="~{index :: head}"></head>
<body>
    <head th:insert="~{index :: navbar}"></head>

    <div class="min-vh-100">
        <div th:if="${successMessage}" class="alert alert-success">
            <p th:text="${successMessage}"></p>
        </div>

        <div class="container">
            <div class="row">
                <div class="col-12 my-5 d-flex justify-content-center align-items-center">
                    <span class="h1">Dashboard Writer</span>
                </div>

                <div class="col-12 my-5 d-flex justify-content-center align-items-center">
                    <span class="h1" th:text="${title}">I tuoi articoli</span>
                </div>


                <div class="col-12">
                    <table class="table table-striped table-responsive-lg">
                        <thead>
                            <tr>
                                <th>Id</th>
                                <th>Title</th>
                                <th>Subtitle</th>
                                <th>Category</th>
                                <th>#Actions</th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr th:each="article:${articles}">
                                <td th:text="${article.id}"></td>
                                <td th:text="${article.title}"></td>
                                <td th:text="${article.subtitle}"></td>
                                <td th:if="${article.category}" th:text="${article.category.name}"></td>
                                <td th:unless="${article.category}">Nessuna categoria</td>
                                <td>
                                    <a th:href="@{/articles/detail/{id}(id=${article.id})}" class="btn btn-primary m-2">
                                        <i class="fa-solid fa-bars"></i>
                                    </a>
                                    <a th:href="@{/articles/edit/{id}(id=${article.id})}" class="btn btn-warning m-2">
                                        <i class="fa-solid fa-pen"></i>
                                    </a>
                                    <a th:href="@{/articles/delete/{id}(id=${article.id})}" class="btn btn-danger m-2">
                                        <i class="fa-solid fa-trash"></i>
                                    </a>
                                </td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
    </div>


    <head th:insert="~{index :: footer}"></head>
    <script th:replace="~{index :: bootstrapScript}"></script>
</body>
</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

funzionante.

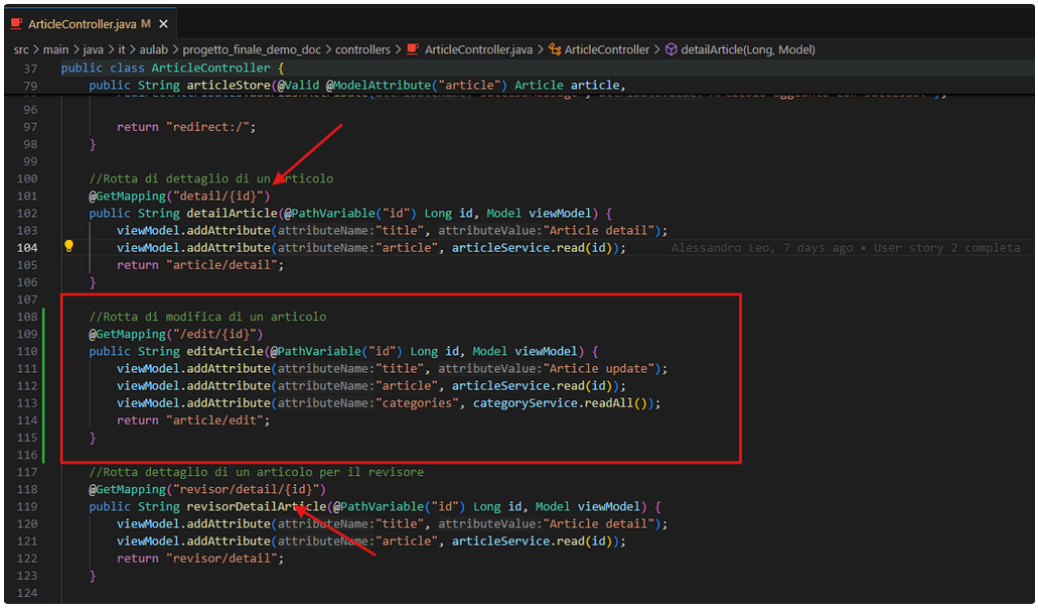
Se provassimo però il tasto di modifica o cancella andrà in errore.

Nessun problema, procediamo con l'implementazione.

EDIT

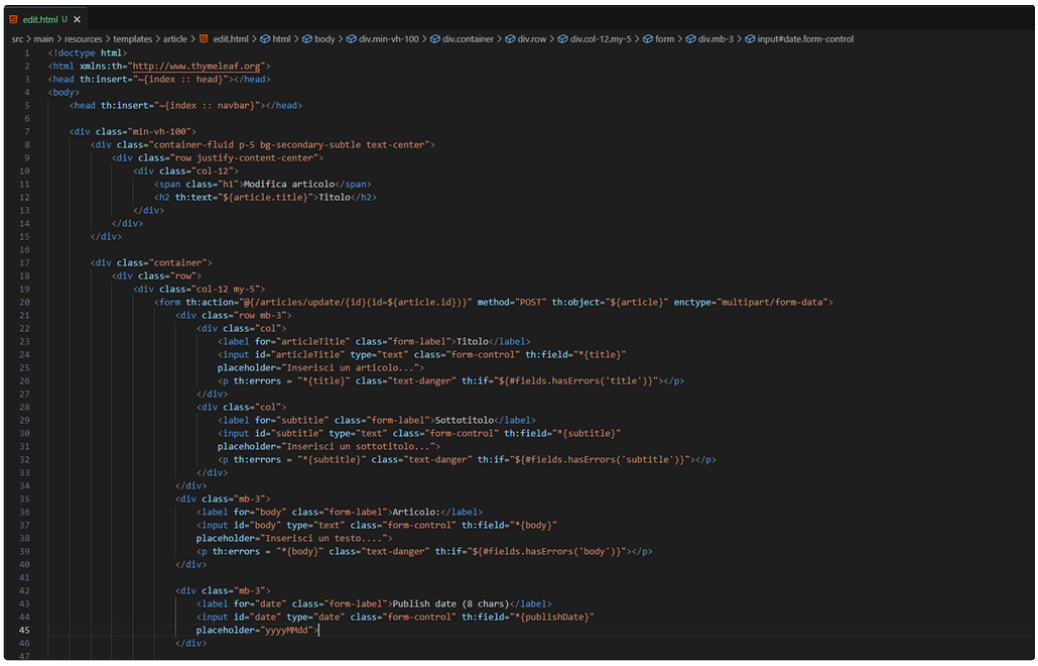
Procediamo con la creazione del meccanismo di modifica. Partiamo sempre dal controller "ArticleController"

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



L'handler appena creato richiamerà il template di modifica che creiamo ora in "src\main\resources\templates\article" e che chiameremo "edit.html"

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



```
                    <div class="mb-3">
                        <label for="category" class="form-label">Category</label>
                        <select id="category" th:field="*{category}" class="form-select">
                            <option th:each="category:${categories}" th:value="${category.id}"
                            th:text="${category.name}"></option>
                            <p th:errors = "*{category}" class="text-danger" th:if="${#fields.hasErrors('category')}"></p>
                        </select>
                    </div>

                    <div class="mb-3 d-flex flex-column" th:if="${article.image != null}">
                        <label for="image" class="form-label">Immagine dell'articolo</label>
                        <img th:src="${article.image.path}" id="image"  alt="${article.image.path}">
                    </div>
                    <div th:unless="${article.image != null}">
                        <label for="image" class="form-label">Immagine dell'articolo</label>
                        <p>Immagine non presente</p>
                    </div>



                    <div class="mb-3">
                        <label for="image" class="form-label">Image</label>
                        <input type="file" name="file" id="image" class="form-control"/>
                    </div>

                    <button type="submit" class="btn btn-success">Modifica articolo</button>
                </form>
            </div>
        </div>
    </div>
</div>


<head th:insert="~{index :: footer}"></head>
<script th:replace="~{index :: bootstrapScript}"></script>
</body>
</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.

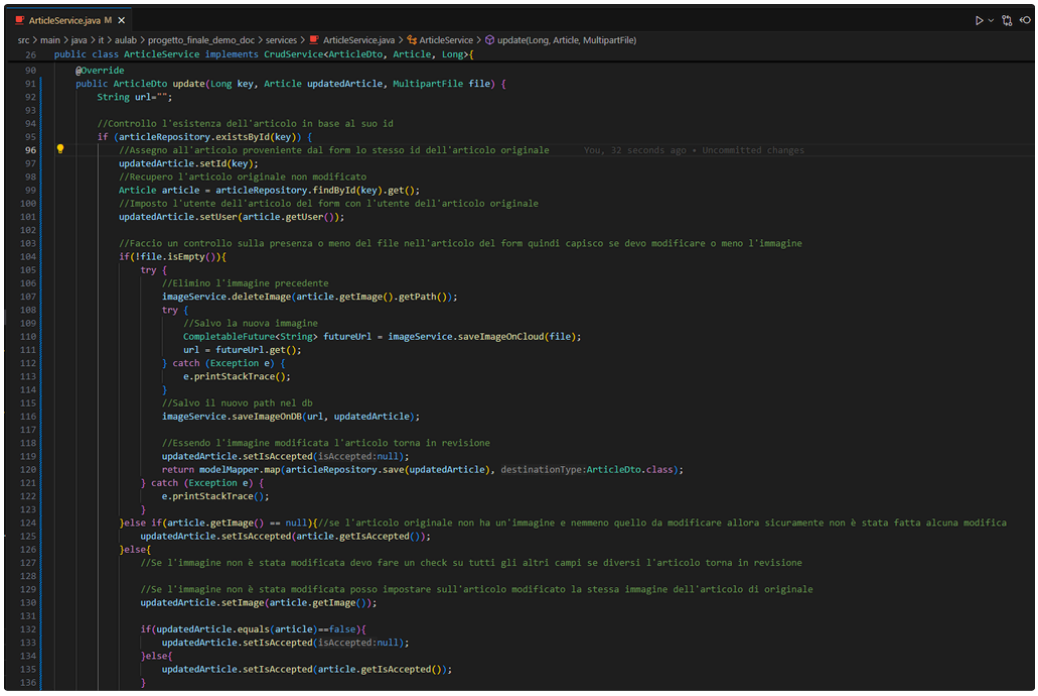
Il form appena creato richiama un handler nel controller non ancora esistente, procediamo con la creazione

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

```
ArticleController.java M
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 {
107
108     //Rotta di modifica di un articolo
109     @GetMapping("/edit/{id}")
110     public String editArticle(@PathVariable("id") Long id, Model viewModel) {
111         viewModel.addAttribute(attributeName:"title", attributeValue:"Article update");
112         viewModel.addAttribute(attributeName:"article", articleService.read(id));
113         viewModel.addAttribute(attributeName:"categories", categoryService.readAll());
114         return "article/edit";
115     }
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,      You, 33 seconds ago • Uncommitted changes
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
```

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"



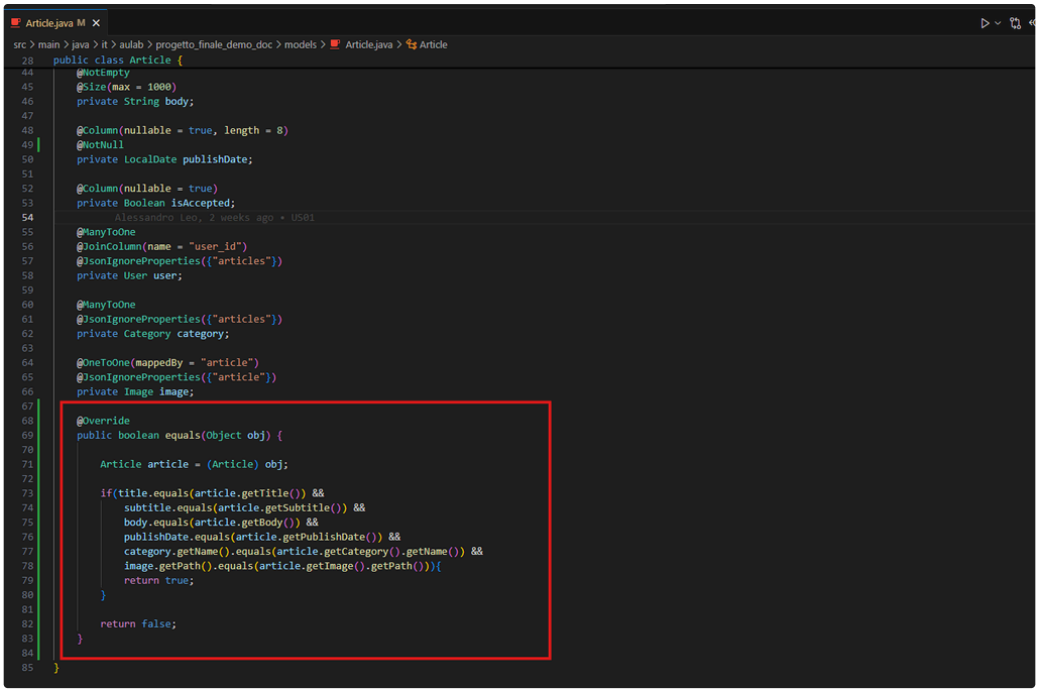
```java
ArticleService.java M
src > main > java > it > aulab > progetto_finale_demo_doc > services > ArticleService.java > ArticleService > update(Long, Article, MultipartFile)
public class ArticleService implements CrudService<ArticleDto, Article, Long>{
    @Override
    public ArticleDto update(Long key, Article updatedArticle, MultipartFile file) {
        String url="";

        //Controllo l'esistenza dell'articolo in base al suo id
        if (articleRepository.existsById(key)) {
            //Assegno all'articolo proveniente dal form lo stesso id dell'articolo originale      You, 32 seconds ago • Uncommitted changes
            updatedArticle.setId(key);
            //Recupero l'articolo originale non modificato
            Article article = articleRepository.findById(key).get();
            //Imposto l'utente dell'articolo del form con l'utente dell'articolo originale
            updatedArticle.setUser(article.getUser());

            //Faccio un controllo sulla presenza o meno del file nell'articolo del form quindi capisco se devo modificare o meno l'immagine
            if(!file.isEmpty()){
                try {
                    //Elimino l'immagine precedente
                    imageService.deleteImage(article.getImage().getPath());
                    try {
                        //Salvo la nuova immagine
                        CompletableFuture<String> futureUrl = imageService.saveImageOnCloud(file);
                        url = futureUrl.get();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    //Salvo il nuovo path nel db
                    imageService.saveImageOnDB(url, updatedArticle);

                    //Essendo l'immagine modificata l'articolo torna in revisione
                    updatedArticle.setIsAccepted(isAccepted:null);
                    return modelMapper.map(articleRepository.save(updatedArticle), destinationType:ArticleDto.class);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }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
                updatedArticle.setIsAccepted(article.getIsAccepted());
            }else{
                //Se l'immagine non è stata modificata devo fare un check su tutti gli altri campi se diversi l'articolo torna in revisione

                //Se l'immagine non è stata modificata posso impostare sull'articolo modificato la stessa immagine dell'articolo di originale
                updatedArticle.setImage(article.getImage());

                if(updatedArticle.equals(article)==false){
                    updatedArticle.setIsAccepted(isAccepted:null);
                }else{
                    updatedArticle.setIsAccepted(article.getIsAccepted());
                }

                return modelMapper.map(articleRepository.save(updatedArticle), destinationType:ArticleDto.class) ;
            }
        } else {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST);
        }
        return null;
    }
```

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“



```java
public class Article {
    @NotEmpty
    @Size(max = 1000)
    private String body;

    @Column(nullable = true, length = 8)
    @NotNull
    private LocalDate publishDate;

    @Column(nullable = true)
    private Boolean isAccepted;
        Alessandro Leo, 2 weeks ago • US01
    @ManyToOne
    @JoinColumn(name = "user_id")
    @JsonIgnoreProperties({"articles"})
    private User user;

    @ManyToOne
    @JsonIgnoreProperties({"articles"})
    private Category category;

    @OneToOne(mappedBy = "article")
    @JsonIgnoreProperties({"article"})
    private Image image;

    @Override
    public boolean equals(Object obj) {

        Article article = (Article) obj;

        if(title.equals(article.getTitle()) &&
            subtitle.equals(article.getSubtitle()) &&
            body.equals(article.getBody()) &&
            publishDate.equals(article.getPublishDate()) &&
            category.getName().equals(article.getCategory().getName()) &&
            image.getPath().equals(article.getImage().getPath())){
            return true;
        }

        return false;
    }
}
```

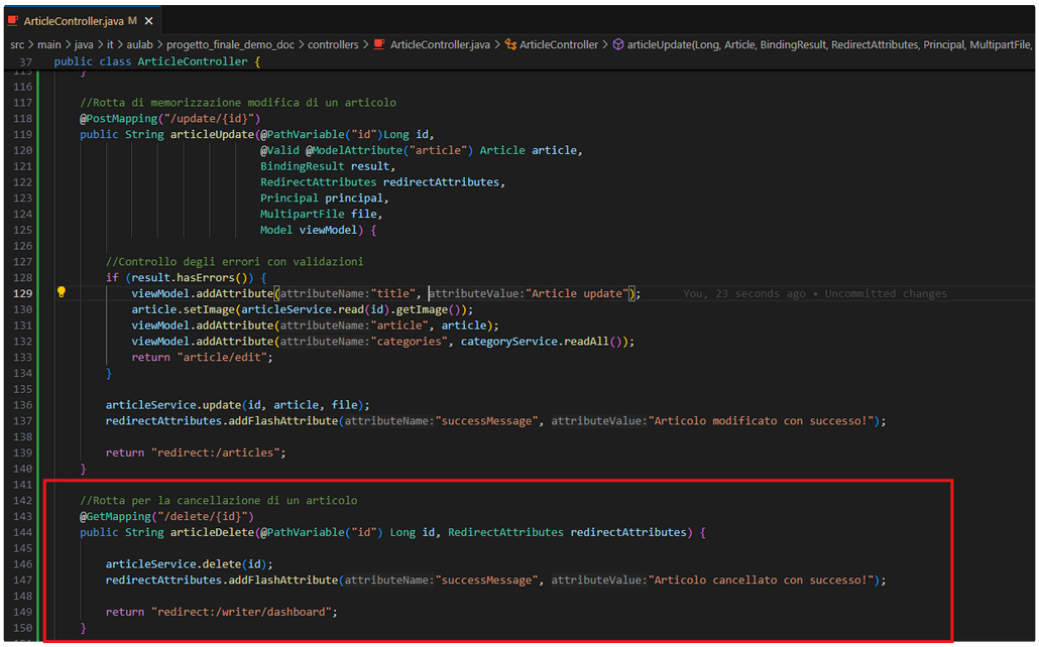
Fatti questi passaggi possiamo procedere con test sulla modifica provando tutte le combinazioni possibili e vedremo che solo se l'articolo è stato modificato tornerà in revisione e le immagini cambiate spariranno dallo storage sostituite da quelle nuove.

DELETE

Procediamo infine con la cancellazione dell'articolo.

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

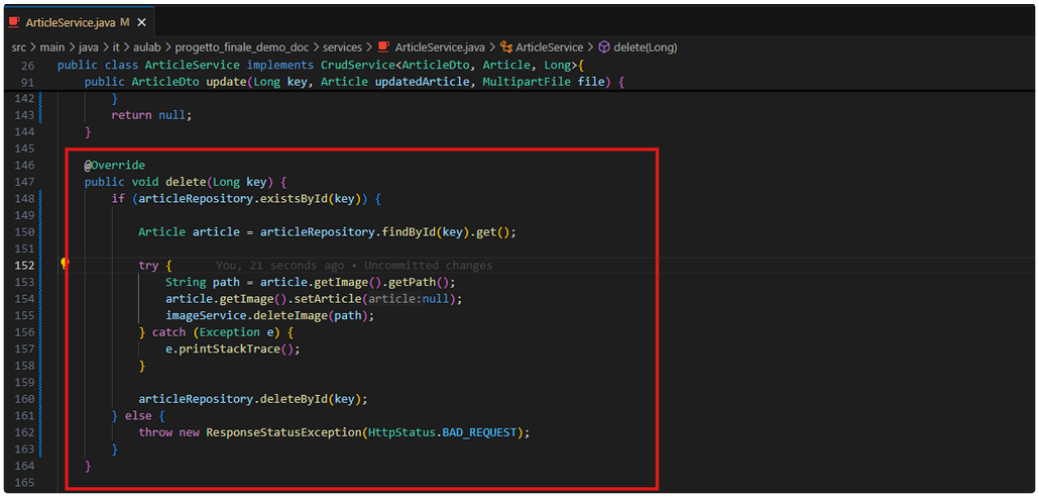
in "src\main\java\it\aulab\progetto_finale_docente\controllers\ArticleController.java"



```java
public class ArticleController {

    //Rotta di memorizzazione modifica di un articolo
    @PostMapping("/update/{id}")
    public String articleUpdate(@PathVariable("id")Long id,
                                @Valid @ModelAttribute("article") Article article,
                                BindingResult result,
                                RedirectAttributes redirectAttributes,
                                Principal principal,
                                MultipartFile file,
                                Model viewModel) {

        //Controllo degli errori con validazioni
        if (result.hasErrors()) {
            viewModel.addAttribute(attributeName:"title", attributeValue:"Article update");      You, 23 seconds ago • Uncommitted changes
            article.setImage(articleService.read(id).getImage());
            viewModel.addAttribute(attributeName:"article", article);
            viewModel.addAttribute(attributeName:"categories", categoryService.readAll());
            return "article/edit";
        }

        articleService.update(id, article, file);
        redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Articolo modificato con successo!");

        return "redirect:/articles";
    }

    //Rotta per la cancellazione di un articolo
    @GetMapping("/delete/{id}")
    public String articleDelete(@PathVariable("id") Long id, RedirectAttributes redirectAttributes) {

        articleService.delete(id);
        redirectAttributes.addFlashAttribute(attributeName:"successMessage", attributeValue:"Articolo cancellato con successo!");

        return "redirect:/writer/dashboard";
    }
```

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“



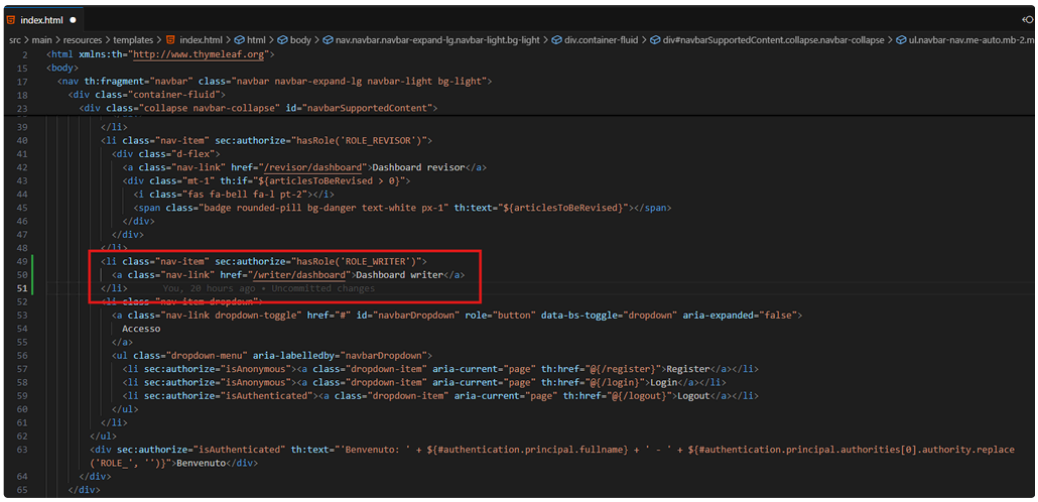
Con questo ultimo passaggio abbiamo completato tutto il CRUD sull’articolo.

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 cosi come abbiamo già fatto per l’admin ed il revisor. Anche questo collegamento faremo in modo che sia visibile solo ad utenti con ruolo wirter.

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



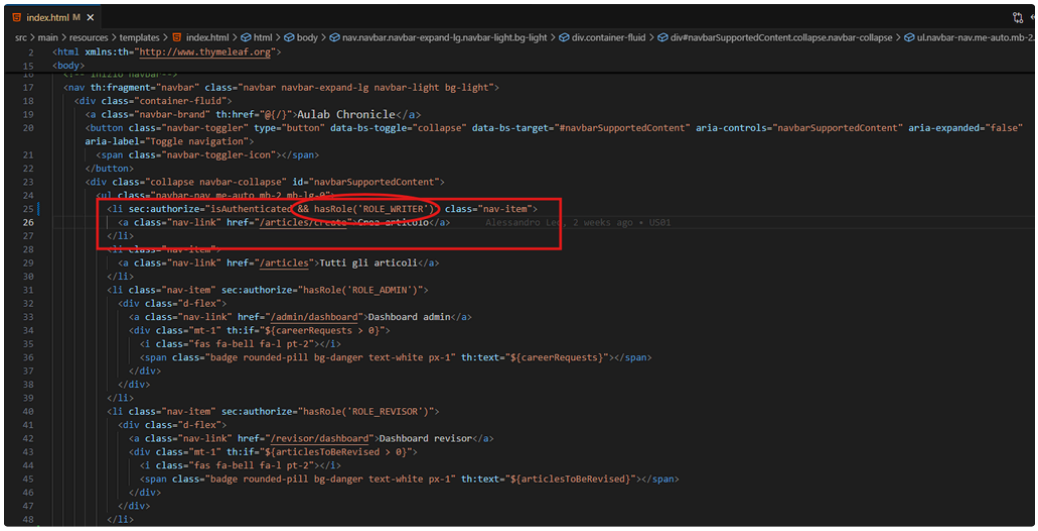
Con questa modifica notiamo immediatamente che il link alla dashboard del writer è disponibile solo a utenti con il ruolo assegnato.

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“



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