

OPIS WSZYSTKICH WYMAGANYCH FUNKCJONALNOŚCI

1. Role Użytkowników

- Admin - zarządza użytkownikami,
- Limited user - użytkownik z ograniczonymi prawami, jest zarejestrowany, może przeglądać udostępnione linki, ale nie może tworzyć informacji,
- Full user - użytkownik z pełnymi prawami, może tworzyć informacje i udostępniać je innym, przeglądać w oddzielnym widoku informacje udostępnione dla niego
- Użytkownik niezalogowany - ma dostęp tylko do strony głównej i strony rejestracji

2. Szczegółowe funkcjonalności, full user

- dodanie/edykcja/usunięcie przez siebie zebranych informacji
- Walidacja formularza
- Edycja na danych bieżących
- Dodanie nowej kategorii
- Wyświetlenie udostępnionych przez innych informacji
- udostępnienie: ze wskazaniem na konkretnego użytkownika lub w linku
- Wyświetlanie "swoich" informacji: sortowanie w obu kierunkach (data,kategoria, alfabetycznie)
- Zapamiętanie kierunków i kryteriów sortowania
- Filtrowanie według daty (od aktualnej) i kategorii (od najbardziej popularnej)
- Logowanie

3. Szczegółowe Funkcjonalności, Niezalogowany (4p)

- Rejestracja
- Walidacja formularza
- Strona powitalna
- Wyświetlenie informacji z udostępnionego linku

4. Szczegółowe Funkcjonalności, admin (2p)

- Wyśw. listy użytkowników
- Zarządzanie rolami

5. Elementy Techniczne (25p.)

- Kontrolery.
- Baza danych (co najmniej 2 tabele z relacją)
- Widoki: formularze z walidacją (3 różne elementy),
- 5 różnych znaczników Thymeleafa.
- Sesja
- Ciasteczka
- Usługa REST (do uwierzytelniania użytkowników).
- Wygląd - (WCAG 2.1)
- Spring Security

IMPLEMENTACJA WYMAGAŃ Z PUNKTU 1 (ROLE)

(Opis szczegółowych funkcjonalności znajduje się w późniejszych fragmentach)

Admin - Admin ze swojego panelu ma możliwość edytowania wszystkich danych użytkowników. Może również usunąć użytkownika z serwisu.

Użytkownik „Limited” - użytkownik z ograniczonymi prawami nie może tworzyć żadnych notatek. Można mu natomiast udostępniać notatki, jeżeli posiada się prawa „Full User” lub „Admin”.

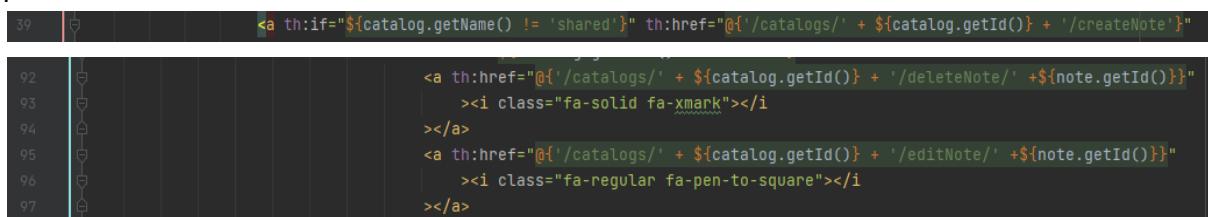
Użytkownik „Full User” - Użytkownik z pełnymi prawami może tworzyć, usuwać i edytować katalogi (foldery z notatkami). W tych katalogach może on tworzyć, usuwać i edytować notatki. Może również udostępniać notatki.

Użytkownik niezalogowany - Dla osoby niezalogowanej jedyną dostępną funkcjonalnością jest możliwość zalogowania się lub utworzenia nowego konta w serwisie.

IMPLEMENTACJA WYMAGAŃ Z PUNKTU 2 (FULL USER)

Dodanie/edykcja/usunięcie przez siebie zebranych informacji

Dodawanie/edykcja i usuwanie notatek możliwe jest dzięki odpowiednim przekierowaniom z pliku **notes.html**



```
39 <a th:if="${catalog.getName() != 'shared'}" th:href="@{'/catalogs/' + ${catalog.getId()} + '/createNote'}">
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
79
80
81
82
83
84
85
86
87
88
89
90
91
92 <a th:href="@{'/catalogs/' + ${catalog.getId()} + '/deleteNote/' + ${note.getId()}}">
93     <i class="fa-solid fa-xmark"></i>
94 </a>
95 <a th:href="@{'/catalogs/' + ${catalog.getId()} + '/editNote/' + ${note.getId()}}">
96     <i class="fa-regular fa-pen-to-square"></i>
97 </a>
```

Przekierowania wyłapywane są przez kontroler **NotesController.java**, który wykonuje odpowiednie operacje.

Tworzenie:

```
309     @GetMapping("createNote")
310     public String redirectCreate(Model model, @PathVariable("catalogId") Integer catalogId, HttpSession session) {
311         Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
312         Integer userId = (Integer) session.getAttribute("userId");
313         if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId) && !(catalog.get().getName().equals("shared"))) {
314             Note note = new Note();
315             model.addAttribute("note", note);
316             model.addAttribute("catalog", catalog.get());
317             return "createNote";
318         }
319         return "redirect:/catalogs";
320     }
321
322     @PostMapping("createNote")
323     public String addNote(@Valid @ModelAttribute("note") Note note, BindingResult bindingResult, Model model, @PathVariable("catalogId") Integer catalogId) {
324         if (bindingResult.hasErrors()) {
325             model.addAttribute("errors", bindingResult);
326             Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
327             model.addAttribute("catalog", catalog.get());
328         }
329         return "createNote";
330     }
```

Usuwanie:

```
335     @GetMapping("deleteNote/{noteId}")
336     public String delete(Model model, @PathVariable("noteId") Integer noteId, @PathVariable("catalogId") Integer catalogId, HttpSession session) {
337         Integer userId = (Integer) session.getAttribute("userId");
338         Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
339         if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId) && !(catalog.get().getName().equals("shared"))) {
340             noteService.deleteNoteById(noteId);
341             return "redirect:/catalogs/" + catalogId;
342         }
343         return "redirect:/catalogs";
344     }
```

Edycja:

```
346     @GetMapping("editNote/{noteId}")
347     public String redirectEdit(Model model, @PathVariable("noteId") Integer noteId, @PathVariable("catalogId") Integer catalogId, HttpSession session) {
348         Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
349         Integer userId = (Integer) session.getAttribute("userId");
350         if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId) && !(catalog.get().getName().equals("shared"))) {
351             Optional<Note> note = noteService.getNoteById(noteId);
352             model.addAttribute("note", note.get());
353             model.addAttribute("catalog", catalog.get());
354             return "editNote";
355         }
356         return "redirect:/catalogs";
357     }
358
359     @PostMapping("editNote/{noteId}")
360     public String editNote(@Valid @ModelAttribute("note") Note note, BindingResult bindingResult, @PathVariable("noteId") Integer noteId, @PathVariable("catalogId") Integer catalogId) {
361         if (bindingResult.hasErrors()) {
362             note.setId(noteId);
363             model.addAttribute("note", note);
364             model.addAttribute("errors", bindingResult);
365             Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
366             model.addAttribute("catalog", catalog.get());
367         }
368         return "editNote";
369     }
```

Metody wyłapujące żądania GET przekierowują odpowiednio na widoki **createNote.html** i **editNote.html**, które po wypełnieniu formularzy przekierowują na odpowiednie linki z metodami POST.

Walidacja formularza

Walidacja została zaimplementowana w klasach, a błędy walidacji są wyświetlane w odpowiednich widokach. Walidacja na przykładzie rejestracji:

Fragment klasy danych **User.java**:

```
31     @Basic
32     @NotBlank(message = "Must not be empty.")
33     @Column(name = "login", nullable = false)
34     @Size(min = 3, max = 20, message = "Size must be between 3 and 20.")
35     @Pattern(regexp = "^[a-z]+$", message = "Login must contain only small letters.")
36     private String login;
37
38     1 usage
39     @Basic
40     @NotBlank(message = "Password is required.")
41     @Size(min = 6, message = "Must be longer than 5 letters.")
42     @Column(name = "password", nullable = false)
43     private String password;
44
45     1 usage
46     @Basic
47     @NotBlank(message = "Must not be empty.")
48     @Column(name = "name", nullable = false)
49     @Size(min = 3, max = 20, message = "Size must be between 3 and 20")
50     @Pattern(regexp = "^[A-Z][a-z]*$", message = "Name must start with capital letter and contain only letters.")
51     private String name;
```

Fragment widoku rejestracji **register.html**:

```
10     <form th:action="@{/register}" th:method="post" th:object="${user}">
11         <h2 class="note">Registration<span class="ally">Form</span></h2>
12         <div class="input-box">
13             <span><i class="fa-sharp fa-solid fa-user"></i></span>
14             <input type="text" id="username" name="username" th:field="#{Login}" th:value="${username}"/>
15             <label for="username">Username<div th:if="${#fields.hasErrors('login') || usernameTaken!=null}" class="error-label-name">
16                 <span th:if="${#fields.hasErrors('login')}" th:errors="*{Login}"></span>
17                 <span th:if="${usernameTaken!=null}" th:text="${usernameTaken}"></span></div></label>
18             </div>
19             <div class="input-box">
20                 <span><i class="fa-sharp fa-solid fa-signature"></i></span>
21                 <input type="text" id="name" name="name" th:field="#{name}" th:value="${name}"/>
22                 <label for="name">Name<div th:if="${#fields.hasErrors('name') || polishNameError!=null}" class="error-label-name">
23                     <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}"></span>
24                     <span th:if="${polishNameError!=null}" th:text="${polishNameError}"></span></div></label>
25                 </div>
26                 <div class="input-box">
27                     <span><i class="fa-sharp fa-solid fa-signature"></i></span>
28                     <input type="text" id="surname" name="surname" th:field="#{surname}" th:value="${surname}"/>
29                     <label for="surname">Surname<div class="error-label-name" th:if="${#fields.hasErrors('surname')}" th:errors="*{surname}"></div></label>
```

Widok w aplikacji przy błędzie walidacji formularza rejestracji:

The screenshot shows a registration form titled "RegistrationForm". The form fields and their current values are:

- Username:** ab
- Name:** aaaaaaaaaaa2
- Surname:** (empty)
- Age:** 13
- Password:** (empty)
- Confirm Password:** (empty)

Validation errors are displayed as blue bars with arrows pointing to specific fields:

- Username:** Size must be between 3 and 20.
- Name:** Name must start with capital letter and contain only letters. Name must be Polish.
- Surname:** Name must start with capital letter and contain only letters. Must not be empty. Size must be between 3 and 50.
- Age:** You must be 18 or older.
- Password:** Must be longer than 5 letters.
- Passwords are different:** Passwords are different.

A large red "Register" button is at the bottom right of the form.

Edycja na danych bieżących

Fragment widoku **editNote.html** odpowiadający za wstawienie bieżących danych przy otwarciu formularza (znaczniki **th:field** importują dane z edytowanej notatki):

```
67 <form>
68     th:action="@{/catalogs/" + ${catalog.id} + '/editNote/' + ${note.getId()}}"
69     th:method="POST"
70     th:object="${note}">
71     <h2 class="note">Edit<span class="ally">Note</span></h2>
72     <div class="input-box">
73         <input type="text" id="title" name="title" th:field="#{title}" />
74         <label for="title">Title<div class="error-label-name" th:if="#{fields.hasErrors('title')}" th:errors="*{title}"></div></label>
75     </div>
76     <div class="input-box">
77         <textarea
78             cols="30"
79             rows="10"
80             type="text"
81             id="content"
82             name="content"
83             th:field="#{content}"></textarea>
84         <label class="content" for="content">Content<div class="error-label-name" th:if="#{fields.hasErrors('content')}" th:errors="*{content}"></div></label>
85     </div>
86     <div class="input-box">
87         <input type="text" id="link" name="link" th:field="#{link}" />
88         <label for="link">Link</label>
```

Dodanie nowej kategorii

Zamiast kategorii notatek, użytkownicy mogą tworzyć katalogi (foldery), w których trzymają swoje notatki. Tytuły katalogów mogą pełnić rolę kategorii, w zależności od potrzeb użytkownika.

Dodawanie/edytacja i usuwanie katalogów możliwe jest dzięki odpowiednim przekierowaniom z widoku **catalogs.html**:

```
27 <div class="add-note anim" style="...">
28     <a th:href="@{/catalogs/' + 'createCatalog'}><i class="fa-solid fa-folder-plus"></i></a>
29 </div>
30
31
32
33
34
35
36
37
38
39
40
41
```

```
54             <div th:if="${catalog.getName() != 'default' && catalog.getName() != 'shared'}">
55                 <a th:href="@{/catalogs/' + 'deleteCatalog/' + ${catalog.getId()}"}>
56                     <i class="fa-solid fa-xmark"></i>
57                 </a>
58
59                 <a th:href="@{/catalogs/' + 'editCatalog/' + ${catalog.getId()}"}>
60                     <i class="fa-regular fa-pen-to-square"></i>
61                 </a>
62             </div>
```

Przekierowania wyłapywane są przez kontroler **CatalogsController.java**, który wykonuje odpowiednie operacje.

Dodawanie:

```
118 @GetMapping("createCatalog")
119 public String redirectCreate(Model model, HttpSession session){
120     Catalog catalog = new Catalog();
121     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
122     model.addAttribute("catalog", catalog);
123     model.addAttribute("user", user.get());
124     return "createCatalog";
125 }
126
127 @PostMapping("createCatalog")
128 public String addCatalog(@Valid @ModelAttribute("catalog") Catalog catalog, BindingResult bindingResult, Model model, HttpSession session) {
129     if (bindingResult.hasErrors()) {
130         model.addAttribute("errors", bindingResult);
131         return "createCatalog";
132     }
133     catalogService.saveCatalog(catalog, (Integer) session.getAttribute("userId"));
134     return "redirect:/catalogs";
135 }
```

Edycja:

```
149 @GetMapping("editCatalog/{catalogId}")
150 public String directEdit(Model model, @PathVariable("catalogId") Integer catalogId, HttpSession session){
151     Integer userId = (Integer)session.getAttribute("userId");
152     Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
153     if(catalog.isPresent() && catalog.get().getUser().getId().equals(userId) && !(catalog.get().getName().equals("default")) &&
154         !(catalog.get().getName().equals("shared")))
155     {
156         model.addAttribute("catalog", catalog.get());
157         return "editCatalog";
158     }
159     return "redirect:/catalogs";
160 }
161
162 @PostMapping("editCatalog/{catalogId}")
163 public String editCatalog(@Valid @ModelAttribute("catalog") Catalog catalog, BindingResult bindingResult,
164                         @PathVariable("catalogId") Integer catalogId, Model model){
165
166     if (bindingResult.hasErrors()) {
167         model.addAttribute("errors", bindingResult);
168         catalog.setId(catalogId);
169         model.addAttribute("catalog", catalog);
170
171         return "editCatalog";
172     }
173     catalogService.updateCatalog(catalog, catalogId);
174     return "redirect:/catalogs";
175 }
```

Usuwanie:

```
137 @GetMapping("deleteCatalog/{catalogId}")
138 public String delete(Model model, @PathVariable("catalogId") Integer catalogId, HttpSession session) {
139     Integer userId = (Integer)session.getAttribute("userId");
140     Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
141     if(catalog.isPresent() && catalog.get().getUser().getId().equals(userId) && !(catalog.get().getName().equals("default"))
142         && !(catalog.get().getName().equals("shared")))
143     {
144         catalogService.deleteCatalogById(catalogId);
145     }
146     return "redirect:/catalogs";
147 }
148 }
```

Metody wyłapujące żądania GET przekierowują odpowiednio na widoki **createCatalog.html** i **editCatalog.html**, które po wypełnieniu formularzy przekierowują na odpowiednie linki z metodami POST.

Wyświetlenie udostępnionych przez innych informacji

Przy rejestracji użytkownika tworzone są dwa unikatowe (nieusuwalne i nieedytowalne) katalogi: **default** i **shared**. Katalog default jest katalogiem domyślnym, a w katalogu **shared** pokazywane są notatki udostępnione przez innych. Udostępnione notatki przechowywane są w bazie danych w osobnej tabeli **shared_notes**, która zawiera id udostępnianej notatki oraz id odbiorcy. Notatki w katalogu **shared** oprócz standardowych atrybutów mają też nazwę osoby udostępniającej.

Udostępnienie: ze wskazaniem na konkretnego użytkownika lub w linku

Zaimplementowaliśmy możliwość udostępnienia notatki innemu użytkownikowi. Po kliknięciu przycisku udostępniania przy notatce, zostajemy przekierowani na widok z formularzem **shareNote.html**, na którym możemy podać użytkownika, któremu chcemy udostępnić notatkę. Pojawi się ona na jego koncie w katalogu **shared**.

Fragment kontrolera **NotesController.java** odpowiadające za udostępnianie:

```
374     @GetMapping("shareNote/{noteId}")
375     public String redirectShare(Model model, @PathVariable("noteId") Integer noteId, @PathVariable("catalogId") Integer catalogId, HttpSession session) {
376         Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
377         Integer userId = (Integer) session.getAttribute("userId");
378         if (catalog.isPresent() && catalog.get().getUser().getUserId().equals(userId) && !(catalog.get().getName().equals("shared"))) {
379             List<User> userList = userService.getUsers();
380             List<User> filteredUserList = new ArrayList<>();
381             Optional<Note> note = noteService.getNoteById(noteId);
382             for (User user : userList) {
383                 if (!user.getUserId().equals(session.getAttribute("userId"))) {
384                     filteredUserList.add(user);
385                 }
386             }
387             model.addAttribute("users", filteredUserList);
388             model.addAttribute("note", note.get());
389             model.addAttribute("catalog", catalog.get());
390             return "shareNote";
391         }
392         return "redirect:/catalogs";
393     }
394
395     @PostMapping("shareNote/{noteId}")
396     public String shareNote(@RequestParam(value = "username") String username, @PathVariable("noteId") Integer noteId,
397                             @PathVariable("catalogId") Integer catalogId, HttpSession session) {
398         Optional<User> user = userService.getUserByUsername(username);
399         if (user.isPresent() && !session.getAttribute("userId").equals(user.get().getId())) {
400             SharedNote sharedNote = new SharedNote();
401             sharedNote.setNote(noteService.getNoteById(noteId).get());
402             sharedNote.setUser(user.get());
403             noteService.saveSharedNote(sharedNote);
404         }
405         return "redirect:/catalogs/" + catalogId;
406     }
}
```

Wyświetlanie “swoich” informacji: sortowanie w obu kierunkach (data, kategoria, alfabetycznie, popularność).

Zaimplementowaliśmy sortowanie katalogów po nazwie oraz popularności (liczbie notatek wewnątrz katalogu) oraz sortowanie notatek po nazwie i dacie. W widoku **catalogs.html** oraz **notes.html** znajdują się odpowiednie linki, dzięki którym możemy uzyskać posortowane informacje.

Fragment widoku katalogów **catalogs.html**:

```
17         <div class="side-title">SORT OPTIONS</div>
18         <div class="side-menu">
19             <a th:href="@{'/catalogs/ASC' }" th:value="1" ><i class="sort fa-solid fa-arrow-down-a-z"></i></a>
20             <a th:href="@{'/catalogs/DESC' }" th:value="2" ><i class="sort fa-solid fa-arrow-up-a-z"></i></a>
21             <a th:href="@{'/catalogs/notesASC' }" th:value="3" ><i class="sort fa-solid fa-arrow-down-1-9"></i></a>
22             <a th:href="@{'/catalogs/notesDESC' }" th:value="4" ><i class="sort fa-solid fa-arrow-up-1-9"></i></a>
23             <a th:href="@{'/catalogs/deleteFilters' }" th:value="5" ><i class="fa-solid fa-filter-circle-xmark"></i></a>
24         </div>
```

Fragment widoku notatek **notes.html**:

```
18         <div class="side-menu">
19             <a th:href="@{'/catalogs/' + ${catalog.getId()}+'/ASC' }" th:value="1" ><i class="sort fa-solid fa-arrow-down-a-z"></i></a>
20             <a th:href="@{'/catalogs/' + ${catalog.getId()}+'/DESC' }" th:value="2" ><i class="sort fa-solid fa-arrow-up-a-z"></i></a>
21             <a th:href="@{'/catalogs/' + ${catalog.getId()}+'/dataASC' }" th:value="3" ><i class="sort fa-regular fa-calendar-plus"></i></a>
22             <a th:href="@{'/catalogs/' + ${catalog.getId()}+'/dataDESC' }" th:value="4" ><i class="sort fa-regular fa-calendar-minus"></i></a>
23             <a th:href="@{'/catalogs/' + ${catalog.getId()}+'/deleteFilters' }" th:value="5" ><i class="fa-solid fa-filter-circle-xmark"></i></a>
```

Fragmenty kontrolera CatalogsController.java:

```
54 @GetMapping("ASC")
55 public String sortCatalogsASC(Model model, HttpSession session, HttpServletRequest request) {
56     Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
57         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).findAny();
58     cookie.get().setValue("ASC");
59     response.addCookie(cookie.get());
60     List<Catalog> catalogList = catalogService.getCatalogsByUserId((Integer) session.getAttribute("userId"));
61     catalogList.sort(Comparator.comparing(Catalog::getName));
62     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
63     model.addAttribute("catalogs", catalogList);
64     model.addAttribute("user", user.get());
65     return "catalogs";
66 }
67 @GetMapping("DESC")
68 public String sortCatalogsDESC(Model model, HttpSession session, HttpServletRequest request, HttpServletResponse response) {
69     Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
70         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).findAny();
71     cookie.get().setValue("DESC");
72     response.addCookie(cookie.get());
73     List<Catalog> catalogList = catalogService.getCatalogsByUserId((Integer) session.getAttribute("userId"));
74     catalogList.sort(Comparator.comparing(Catalog::getName).reversed());
75     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
76     model.addAttribute("catalogs", catalogList);
77     model.addAttribute("user", user.get());
78     return "catalogs";
79 }

81 @GetMapping("/notesASC")
82 public String sortCatalogsByNotesASC(Model model, HttpSession session, HttpServletRequest request, HttpServletResponse response) {
83     Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
84         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).findAny();
85     cookie.get().setValue("notesASC");
86     response.addCookie(cookie.get());
87     List<Catalog> catalogList = catalogService.getCatalogsByUserId((Integer) session.getAttribute("userId"));
88     catalogList.sort(Comparator.comparing(Catalog::getNoteCount));
89     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
90     model.addAttribute("catalogs", catalogList);
91     model.addAttribute("user", user.get());
92     return "catalogs";
93 }

95 @GetMapping("/notesDESC")
96 public String sortCatalogsByNotesDESC(Model model, HttpSession session, HttpServletRequest request, HttpServletResponse response) {
97     Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
98         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).findAny();
99     cookie.get().setValue("notesDESC");
100    response.addCookie(cookie.get());
101    List<Catalog> catalogList = catalogService.getCatalogsByUserId((Integer) session.getAttribute("userId"));
102    catalogList.sort(Comparator.comparing(Catalog::getNoteCount).reversed());
103    Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
104    model.addAttribute("catalogs", catalogList);
105    model.addAttribute("user", user.get());
106    return "catalogs";
107 }

109 @GetMapping("deleteFilters")
110 public String deleteFilters(Model model, HttpSession session, HttpServletRequest request, HttpServletResponse response) {
111     Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
112         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).findAny();
113     cookie.get().setValue("default");
114     response.addCookie(cookie.get());
115     return "redirect:/catalogs";
116 }
```

Fragmenty kontrolera NotesController.java:

```
89  @GetMapping("/*/{catalogId}")
90  public String sortNotesByTitleASC(Model model, @PathVariable("catalogId") Integer catalogId,
91                                     HttpSession session, HttpServletResponse response, HttpServletRequest request) {
92
93      Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
94      Integer userId = (Integer) session.getAttribute("userId");
95
96      if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId)) {
97          Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
98              c -> c.getName().equals("noteCookie" + session.getAttribute("userId"))).findAny();
99          cookie.get().setValue("ASC");
100         response.addCookie(cookie.get());
101
102     List<Note> noteList;
103     if (catalog.get().getName().equals("shared")) {
104         noteList = noteService.getNotesFromSharedByCatalogId(userId);
105     } else {
106         noteList = noteService.getNotesByCatalogId(catalogId);
107     }
108     LocalDate oldDate = null;
109     LocalDate newDate = null;
110
111     for (Note n : noteList) {
112         LocalDate currentDate = n.getDate();
113
114         if (oldDate == null || currentDate.isBefore(oldDate)) {
115             oldDate = currentDate;
116         }
117
118         if (newDate == null || currentDate.isAfter(newDate)) {
119             newDate = currentDate;
120         }
121     }
122
123     if (oldDate == null && newDate == null) {
124         oldDate = LocalDate.now();
125         newDate = LocalDate.now();
126     }
127
128     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
129     noteList.sort(Comparator.comparing(Note::getTitle));
130     model.addAttribute("oldDate", oldDate);
131     model.addAttribute("newDate", newDate);
132     model.addAttribute("sharedNotes", noteService.getMySharedNotes(userId));
133     model.addAttribute("catalog", catalog.get());
134     model.addAttribute("notes", noteList);
135     model.addAttribute("user", user.get());
136     return "notes";
137 }
138
139 }
140
141 @GetMapping("/*/{catalogId}")
142 public String sortNotesByTitleDESC(Model model, @PathVariable("catalogId") Integer catalogId,
143                                     HttpSession session, HttpServletResponse response, HttpServletRequest request) {
144     Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
145     Integer userId = (Integer) session.getAttribute("userId");
146
147     if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId)) {
148         Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
149             c -> c.getName().equals("noteCookie" + session.getAttribute("userId"))).findAny();
150         cookie.get().setValue("DESC");
151         response.addCookie(cookie.get());
152
153         List<Note> noteList;
154         if (catalog.get().getName().equals("shared")) {
155             noteList = noteService.getNotesFromSharedByCatalogId(userId);
156         } else {
157             noteList = noteService.getNotesByCatalogId(catalogId);
158         }
159
160         LocalDate oldDate = null;
161         LocalDate newDate = null;
```

```

162
163     for (Note n : noteList) {
164         LocalDate currentDate = n.getDate();
165
166         if (oldDate == null || currentDate.isBefore(oldDate)) {
167             oldDate = currentDate;
168         }
169
170         if (newDate == null || currentDate.isAfter(newDate)) {
171             newDate = currentDate;
172         }
173     }
174
175     if(oldDate == null && newDate == null){
176         oldDate = LocalDate.now();
177         newDate = LocalDate.now();
178     }
179
180     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
181     noteList.sort(Comparator.comparing(Note::getTitle).reversed());
182     model.addAttribute("oldDate", oldDate);
183     model.addAttribute("newDate", newDate);
184     model.addAttribute("sharedNotes", noteService.getMySharedNotes(userId));
185     model.addAttribute("catalog", catalog.get());
186     model.addAttribute("notes", noteList);
187     model.addAttribute("user", user.get());
188     return "notes";
189 }
190
191
192
193 @GetMapping("/dataASC")
194 public String sortNotesByDateASC(Model model, @PathVariable("catalogId") Integer catalogId,
195                                     HttpSession session, HttpServletResponse response, HttpServletRequest request) {
196     Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
197     Integer userId = (Integer) session.getAttribute("userId");
198
199     if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId)) {
200         Optional<Cookie> cookie = Arrays.stream(request.getCookies()).filter(
201             c -> c.getName().equals("noteCookie" + session.getAttribute("userId")).findAny();
202         cookie.get().setValue("dataASC");
203         response.addCookie(cookie.get());
204         List<Note> noteList;
205         if (catalog.get().getName().equals("shared")) {
206             noteList = noteService.getNotesFromSharedByCatalogId(userId);
207         } else {
208             noteList = noteService.getNotesByCatalogId(catalogId);
209         }
210         LocalDate oldDate = null;
211         LocalDate newDate = null;
212         for (Note n : noteList) {
213             LocalDate currentDate = n.getDate();
214             if (oldDate == null || currentDate.isBefore(oldDate)) {
215                 oldDate = currentDate;
216             }
217
218             if (newDate == null || currentDate.isAfter(newDate)) {
219                 newDate = currentDate;
220             }
221         }
222         if(oldDate == null && newDate == null){
223             oldDate = LocalDate.now();
224             newDate = LocalDate.now();
225         }
226         Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
227         noteList.sort(Comparator.comparing(Note::getDate));
228         model.addAttribute("oldDate", oldDate);
229         model.addAttribute("newDate", newDate);
230         model.addAttribute("sharedNotes", noteService.getMySharedNotes(userId));
231         model.addAttribute("catalog", catalog.get());
232         model.addAttribute("notes", noteList);
233         model.addAttribute("user", user.get());
234         return "notes";
235     }
236
237     return "redirect:/catalogs";

```

Zapamiętanie kierunków i kryteriów sortowania

Do zapamiętywania kierunków sortowania katalogów oraz notatek używamy ciasteczek. Tworzone są one przy rejestracji, a ustawiane podczas wyboru kierunku sortowania (co widać na zdjęciach w punkcie powyżej). Kontrolery **NotesController.java** oraz **CatalogsController.java** odczytują te ciasteczna i przekierowują na odpowiednie linki.

Fragment kontrolera **CatalogsController.java**:

```
31     @GetMapping("/*")
32     public String getCatalogsById(Model model, HttpSession session, HttpServletRequest request) {
33         Optional<String> sortValue = Arrays.stream(request.getCookies()).filter(
34             c -> c.getName().equals("catalogCookie") + session.getAttribute("userId")).map(Cookie::getValue).findAny();
35
36         if(sortValue.isEmpty())
37             return "redirect:/login";
38
39         if (sortValue.get().equals("ASC"))
40             return "redirect:/catalogs/ASC";
41         else if (sortValue.get().equals("DESC"))
42             return "redirect:/catalogs/DESC";
43         else if (sortValue.get().equals("notesASC"))
44             return "redirect:/catalogs/notesASC";
45         else if (sortValue.get().equals("notesDESC"))
46             return "redirect:/catalogs/notesDESC";
47     }
```

Fragment kontrolera **NotesController.java**:

```
31     @GetMapping("/*")
32     public String getNotesByCatalogId(Model model, @PathVariable("catalogId") Integer catalogId, HttpSession session, HttpServletRequest request) {
33         Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
34         Integer userId = (Integer) session.getAttribute("userId");
35
36         Optional<String> sortValue = Arrays.stream(request.getCookies()).filter(
37             c -> c.getName().equals("noteCookie") + session.getAttribute("userId")).map(Cookie::getValue).findAny();
38
39         if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId)) {
40             if (sortValue.get().equals("ASC"))
41                 return "redirect:/catalogs/{catalogId}/ASC";
42             else if (sortValue.get().equals("DESC"))
43                 return "redirect:/catalogs/{catalogId}/DESC";
44             else if (sortValue.get().equals("dataASC"))
45                 return "redirect:/catalogs/{catalogId}/dataASC";
46             else if (sortValue.get().equals("dataDESC"))
47                 return "redirect:/catalogs/{catalogId}/dataDESC";
48         }
49     }
```

Filtrowanie

Zaimplementowaliśmy filtrowanie notatek po dacie. Użytkownik może wybrać datę początkową i końcową, następnie następuje przekierowanie na link, gdzie lista notatek zostaje przefiltrowana oraz pokazana. Został również utworzony pasek wyszukiwania pozwalający znajdować notatki lub katalogi po nazwie.

Fragment widoku **notes.html** z formularzem do wyboru daty:

```
28         <form class="dates" th:action="@{/catalogs/" + ${catalog.getId()} + "/filterByDates}" th:method="post">
29             <label class="label-date">
30                 <span class="date-note">From: </span>
31                 <input type="date" name="startDate" th:value="${oldDate}" />
32             </label>
33             <label class="label-date">
34                 <span class="date-note">To: </span>
35                 <input type="date" name="endDate" th:value="${newDate}" />
36             </label>
37             <button class="filter" type="submit"><i class="fa-solid fa-filter" ></i></button>
38         </form>
39         <a th:if="${catalog.getName() != 'shared'}" th:href="@{/catalogs/" + ${catalog.getId()} + '/createNote'}"
40             ><i class="fa-regular fa-square-plus"></i>
41         </a>
```

Fragment kontrolera NotesController.java:

```
468 @PostMapping(value="/filterByDates")
469 public String filter(@RequestParam(name="startDate")
470                      LocalDate startDate,
471                      @RequestParam(name="endDate") LocalDate endDate, Model model, @PathVariable("catalogId") Integer catalogId, HttpSession session){
472
473     Optional<Catalog> catalog = catalogService.getCatalogById(catalogId);
474     Integer userId = (Integer) session.getAttribute("userId");
475     if (catalog.isPresent() && catalog.get().getUser().getId().equals(userId)) {
476         List<Note> noteList;
477         if (catalog.get().getName().equals("shared")) {
478             noteList = noteService.getNotesFromSharedByCatalogId(userId);
479         } else {
480             noteList = noteService.getNotesByCatalogId(catalogId);
481         }
482         Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
483         model.addAttribute("oldDate", startDate);
484         model.addAttribute("newDate", endDate);
485         model.addAttribute("user", user.get());
486         model.addAttribute("sharedNotes", noteService.getMySharedNotes(userId));
487         model.addAttribute("catalog", catalog.get());model.addAttribute("notes", noteList.stream().filter(n
488             -> (n.getDate().isAfter(startDate) && n.getDate().isBefore(endDate))
489             || n.getDate().isEqual(startDate) || n.getDate().isEqual(endDate)).toList());
490     }
491     return "notes";
492 }
493 }
```

Logowanie

Logowanie jest oparte na formularzu zawartym w **login.html**. Autentykacją użytkownika zajmuje się Spring Security, który został przez nas odpowiednio skonfigurowany.

Fragment widoku login.html:

```
17 <div class="login-box">
18     <form th:action="@{/login}" method="post" role="form">
19         <h2 class="note">Note<span class="ally">ally</span></h2>
20         <div class="input-box">
21             <span><i class="fa-sharp fa-solid fa-user"></i></span>
22             <input type="text" id="login" name="login" class="form-control" />
23             <label for="login">Login<div class="error-label-name" th:if="${session.info!=null
24                 && !session.info.equals('')}" th:text="${session.info}"></div></label>
25         </div>
```

Fragment Log_Reg_Controller.java:

```
26
27     @GetMapping(value="/login")
28     public String redirectLogin(HttpSession session){
29         if(session.getAttribute("userId") != null)
30         {
31             return "redirect:/catalogs";
32         }
33         return "login";
34     }
35 }
```

Fragment pliku konfiguracyjnego modułu Spring Security **WebSecurity.java**:

```
43     .formLogin()
44         form -> form
45             .loginPage("/login")
46             .usernameParameter("login")
47             .passwordParameter("password")
48             .loginProcessingUrl("/Login")
49             .permitAll()
50             .defaultSuccessUrl("/catalogs")
51             .failureHandler(new AuthenticationFailureHandler() {
52                 @Override
53                 public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception)
54                     throws IOException, ServletException {
55                         HttpSession session = request.getSession();
56                         session.setAttribute("info", "Incorrect Login or Password");
57                         response.sendRedirect("/login");
58                     }
59             })
60         .sessionManagement(
61             session -> session
62                 .sessionCreationPolicy(SessionCreationPolicy.ALWAYS) SessionManagementConfigurer<HttpSecurity>
63                 .invalidSessionUrl("/Login")
64                 .maximumSessions(1) ConcurrencyControlConfigurer
65                 .maxSessionsPreventsLogin(false)
66         }
```

Dodatkowo został przez nas zaimplementowany dodatkowy “listener” **AuthenticationApplicationListener.java**, który przy udanej autentykacji dodaje do sesji atrybut **userId**, który jest potem wykorzystywany np. do pobrania odpowiednich katalogów po id użytkownika.

AuthenticationApplicationListener.java:

```
1 package pl.noteally.config;
2
3
4 import jakarta.servlet.http.HttpSession;
5 import lombok.AllArgsConstructor;
6 import org.springframework.context.event.EventListener;
7 import org.springframework.security.authentication.event.InteractiveAuthenticationSuccessEvent;
8 import org.springframework.stereotype.Component;
9 import pl.noteally.data.User;
10
11 @Component
12 @AllArgsConstructor
13 public class AuthenticationApplicationListener {
14
15     private final HttpSession session;
16     @EventListener
17     @HandlesEvent(InteractiveAuthenticationSuccessEvent.class)
18     public void handleInteractiveAuthenticationSuccess(InteractiveAuthenticationSuccessEvent event) {
19         User user = (User) event.getAuthentication().getPrincipal();
20         session.setAttribute("userId", user.getId());
21     }
}
```

IMPLEMENTACJA WYMAGAŃ UŻYTKOWNIKA LIMITED

W klasie **WebSecurity.java** są zdefiniowane odpowiednie uprawnienia dla poszczególnych użytkowników. Użytkownik o roli "LIMITED USER" ma ograniczone uprawnienia dostępu, które nie obejmują tworzenia ani udostępniania notatek.

Fragment pliku konfiguracyjnego **WebSecurity.java**:

```
33  @Bean
34  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
35      http
36          .authorizeHttpRequests(authz -> authz
37              .requestMatchers("/css/**", "/images/**", "/js/**").permitAll()
38              .requestMatchers("/register", "/", "/login").permitAll()
39              .requestMatchers("/admin/**").hasAuthority("ADMIN")
40              .requestMatchers("/catalogs/createCatalog", "/catalogs/deleteCatalog", "/catalogs/editCatalog").
41              hasAnyAuthority(...authorities: "ADMIN", "USER")
42              .requestMatchers("/catalogs/{catalogId}/*").hasAnyAuthority(...authorities: "ADMIN", "USER")
43              .anyRequest().authenticated()
```

IMPLEMENTACJA WYMAGAŃ Z PUNKTU 3 (NIEZALOGOWANY)

Rejestracja

Rejestracja odbywa się za pomocą formularza w widoku **register.html**, który tworzy obiekt **User** i przesyła go na odpowiedni link z metodą POST. W kontrolerze **Log_Reg_Controller.java** odbywa się również sprawdzanie błędów walidacji oraz zapytanie do utworzonego przez nas serwera REST o nazwie **isNamePolish**, który sprawdza, czy podane imię jest polskie.

Fragment formularza w widoku **register.html**:

```
14  <form th:action="@{/register}" th:method="post" th:object="${user}">
15      <h2 class="note">Registration</h2>
16      <div class="input-box">
17          <span><i class="fa-sharp fa-solid fa-user"></i></span>
18          <input type="text" id="username" name="username" th:field="#{login}" th:value="${username}">
19          <label for="username">Username</label>
20          <div th:if="#{fields.hasErrors('login') || usernameTaken!=null}" class="error-label-name">
21              <span th:if="#{usernameTaken!=null}" th:text="${usernameTaken}"></span></div>
22      </div>
23      <div class="input-box">
24          <span><i class="fa-sharp fa-solid fa-signature"></i></span>
25          <input type="text" id="name" name="name" th:field="#{name}" th:value="${name}">
26          <label for="name">Name</label>
27          <div th:if="#{fields.hasErrors('name') || polishNameError!=null}" class="error-label-name">
28              <span th:if="#{fields.hasErrors('name')}" th:text="${name}"></span>
29              <span th:if="#{polishNameError!=null}" th:text="${polishNameError}"></span></div>
30      </div>
31      <div class="input-box">
32          <span><i class="fa-sharp fa-solid fa-signature"></i></span>
33          <input type="text" id="surname" name="surname" th:field="#{surname}" th:value="${surname}">
34          <label for="surname">Surname</label>
35          <div class="error-label-name" th:if="#{fields.hasErrors('surname')}" th:errors="*{surname}"></div>
36      </div>
37      <div class="input-box">
38          <span><i class="fa-sharp fa-solid fa-calendar"></i></span>
39          <input type="number" id="age" name="age" th:field="#{age}" th:value="${age}">
40          <label for="age">Age</label>
41          <div class="error-label-name" th:if="#{fields.hasErrors('age')}" th:errors="*{age}"></div>
42      </div>
43      <div class="input-box">
44          <span><i class="fa-sharp fa-solid fa-lock"></i></span>
45          <input type="password" id="password" name="password" th:field="#{password}" th:value="${password}">
46          <label for="password">Password</label>
47          <div class="error-label-name" th:if="#{fields.hasErrors('password')}" th:errors="*{password}"></div>
```

Fragmenty kontrolera Log_Reg_Controller.java:

```
36 @GetMapping("/register")
37     public String redirectRegister(Model model, HttpSession session){
38         if(session.getAttribute("userId") != null)
39         {
40             return "redirect:/catalogs";
41         }
42
43         model.addAttribute("attributeName: "user", new User());
44         return "register";
45     }
46
47 @PostMapping("/register")
48     public String register(@Valid @ModelAttribute("user") User user, BindingResult bindingResult, @RequestParam ("confirmPassword") String confirmPassword
49     HttpServletResponse response, Model model) {
50         boolean errors = false;
51         model.addAttribute("user", user);
52         model.addAttribute("username", user.getUsername());
53         model.addAttribute("name", user.getName());
54         model.addAttribute("surname", user.getSurname());
55         model.addAttribute("age", user.getAge());
56         model.addAttribute("password", user.getPassword());
57
58         if(userService.userExists(user)) {
59             model.addAttribute("usernameTaken", "This username is already taken");
60             errors = true;
61         }
62         if(!user.getPassword().equals(confirmPassword)) {
63             model.addAttribute("confirmError", "Passwords are different");
64             errors = true;
65         }
66         if(!restNameService.isNamePolish(user.getName())) {
67             model.addAttribute("polishNameError", "Name must be Polish");
68             errors = true;
69         }
70         if(bindingResult.hasErrors()) {
71             model.addAttribute("errors", bindingResult);
72             errors = true;
73         }
74         if(errors) return "register";
75
76         userService.signUpUser(user);
77
78         Cookie catalogCookie = new Cookie("catalogCookie" + user.getId(), "default");
79         catalogCookie.setMaxAge(60 * 60 * 24 * 365 * 10);
80         Cookie noteCookie = new Cookie("noteCookie" + user.getId(), "default");
81         noteCookie.setMaxAge(60 * 60 * 24 * 365 * 10);
82         response.addCookie(catalogCookie);
83         response.addCookie(noteCookie);
84
85         return "login";
```

Fragment serwisu **UserService.java**:

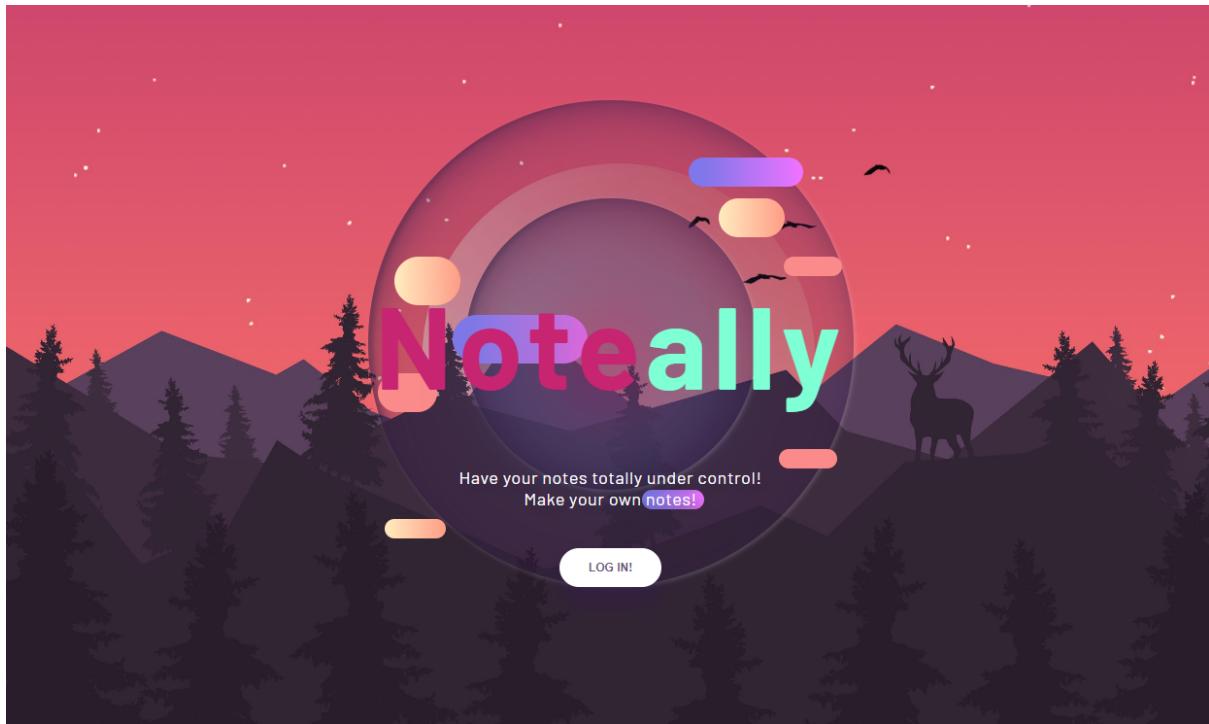
```
38 @ ... public String signUpUser(User user){  
39  
40     String encodedPassword = bCryptPasswordEncoder.encode(user.getPassword());  
41  
42     user.setPassword(encodedPassword);  
43  
44     userRepository.save(user);  
45  
46     //default catalogs  
47     Catalog defaultCatalog = new Catalog();  
48     defaultCatalog.setName("default");  
49     defaultCatalog.setUser(user);  
50     catalogRepository.save(defaultCatalog);  
51  
52     Catalog sharedCatalog = new Catalog();  
53     sharedCatalog.setName("shared");  
54     sharedCatalog.setUser(user);  
55     catalogRepository.save(sharedCatalog);  
56  
57     return "catalogs";  
58 }
```

Walidacja formularza

Walidacja formularza rejestracji została opisana wcześniej (punkt "Walidacja Formularza" przy opisywaniu użytkownika z pełnymi prawami)

Strona powitalna

Widok strony powitalnej jest dostępny dla każdego użytkownika:



Wyświetlenie informacji z udostępnionego linku

Nie zaimplementowaliśmy udostępniania notatek przez link, ponieważ kłociło się to z naszą wizją projektu. Notatkę możemy udostępnić tylko innemu, zalogowanemu użytkownikowi.

IMPLEMENTACJA WYMAGAŃ Z PUNKTU 4 (ADMIN)

Wyśw. listy użytkowników

W panelu Admina, do którego dostęp ma tylko użytkownik o roli “ADMIN”, wyświetla się lista użytkowników (można ją sortować alfabetycznie). Admin może edytować i usuwać użytkowników.

Fragment widoku **admin.html**:

```
36 <ul th:each="user : ${userList}">
37     <div class="container-user">
38         <li>
39             <i class="fa-solid fa-user"></i>
40             <p>
41                 th:href="@{'/admin/' + ${user.getId()}}"
42                 th:text="${user.getLogin()}"
43                 >Username</p>
44             >
45             </li>
46             <a th:href="@{'/admin/' + 'deleteUser/' + ${user.getId()}}"
47                 ><i class="fa-solid fa-xmark"></i>
48             ></a>
49
50             <a th:href="@{'/admin/' + 'editUser/' + ${user.getId()}}"
51                 ><i class="fa-regular fa-pen-to-square"></i>
52             ></a>
53         </div>
54     </ul>
```

Fragment kontrolera **AdminController.java**:

```
24 @GetMapping("admin")
25 public String getAllUser(Model model, HttpSession session) {
26     List<User> userList=userService.getUsers();
27     List<User> filteredUserList = new ArrayList<>();
28     for (User user : userList) {
29         if (!user.getId().equals(session.getAttribute("userId"))){
30             filteredUserList.add(user);
31         }
32     }
33     model.addAttribute("userList", filteredUserList);
34     return "admin";
35 }
36 }
```

Zarządzanie rolami

Admin może edytować wszystkie dane użytkowników (w tym rolę). Po wybraniu przycisku edycji przy użytkowniku, zostaje przekierowany na formularz edycji **editUser.html**.

Fragment widoku **editUser.html** odpowiedzialny za wybór nowej roli użytkownika:

```
81 <div class="input-box">
82     <select id="role" name="role" th:field="*{role}">
83         <option value="ADMIN">ADMIN</option>
84         <option value="USER">USER</option>
85         <option value="LIMITED">LIMITED</option>
86     </select>
87     <label for="role">Role</label>
88 </div>
```

Fragment kontrolera **AdminController.java**:

```
64 @GetMapping("/deleteUser/{userId}")
65 public String deleteUser(Model model, @PathVariable("userId") Integer userId, HttpSession session) {
66     if((session.getAttribute("userId").equals(userId))
67     {
68         return "redirect:/admin";
69     }
70     userService.deleteUserById(userId);
71     return "redirect:/admin";
72 }
73 @GetMapping("/editUser/{userId}")
74 public String redirectEdit(Model model, @PathVariable("userId") Integer userId, HttpSession session){
75     if((session.getAttribute("userId").equals(userId))
76     {
77         return "redirect:/admin";
78     }
79     Optional<User> user = userService.getUserById(userId);
80     model.addAttribute( attributeName: "user", user.get());
81     return "editUser";
82 }
83
84 @PostMapping("/editUser/{userId}")
85 public String editUser(@Valid @ModelAttribute("user") User user, BindingResult bindingResult, @PathVariable("userId") Integer userId, Model model){
86
87     if (bindingResult.hasErrors()) {
88         user.setId(userId);
89         model.addAttribute( attributeName: "user", user);
90         model.addAttribute( attributeName: "errors",bindingResult);
91         return "editUser";
92     }
93     userService.updateUser(user, userId);
94     return "redirect:/admin";
95 }
96 }
```

Fragment serwisu **UserService.java**:

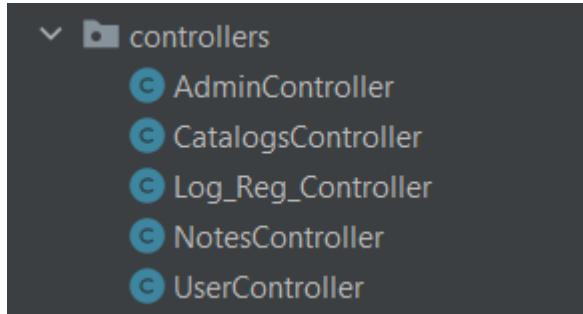
```
70 @
71     public void updateUser(User user, Integer userId)
72     {
73         Optional<User> existingUser = userRepository.findById(userId);
74         existingUser.get().setLogin(user.getLogin());
75         existingUser.get().setName(user.getName());
76         existingUser.get().setSurname(user.getSurname());
77         existingUser.get().setAge(user.getAge());
78         existingUser.get().setRole(user.getRole());
79         userRepository.save(existingUser.get());
80     }
```

IMPLEMENTACJA WYMAGAŃ Z PUNKTU 5 (ELEMENTY TECHNICZNE)

Kontrolery

We wcześniejszych punktach dokumentacji opisano działanie poszczególnych kontrolerów, które obsługują określone żądania użytkowników.

Lista utworzonych przez nas kontrolerów:



5 różnych znaczników Thymeleafa

Lista wszystkich znaczników Thymeleafa użytych w projekcie:

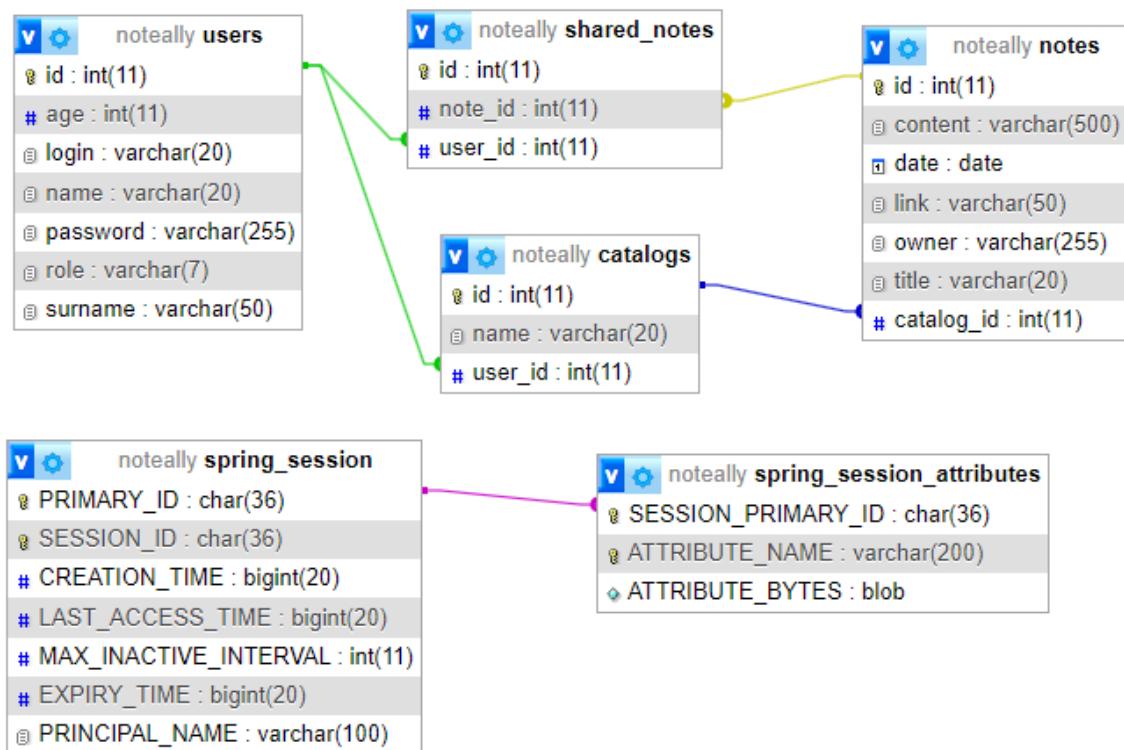
- th:href
- th:each
- th:if
- th:field
- th:src
- th:method
- th:object
- th:text
- th:value

Wykorzystanie thymeleafa w plikach .html umożliwiło nam dynamiczną interakcję frontendu z backendem.

Baza danych (co najmniej 2 tabele z relacją)

Do wykonania projektu wykorzystaliśmy bazę danych **MySQL**. Posiada ona 4 tabele utworzone na podstawie klas danych w projekcie (**users**, **catalogs**, **notes**, **shared_notes**) oraz 2 tabele tworzone przez Springa do zarządzania sesją użytkownika (**spring_session**, **spring_session_attributes**).

Schemat bazy danych z panelu **phpMyAdmin**:



Widoki: formularze z walidacją (3 różne elementy)

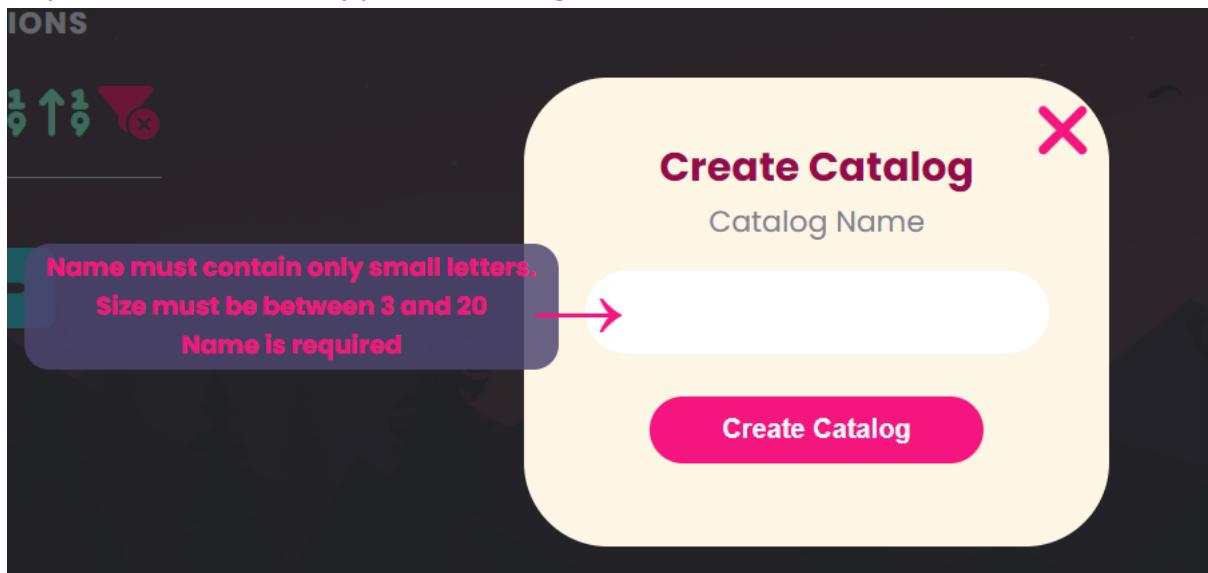
Przykład widoku z walidacją `login.html`:

The screenshot shows the login interface for the Noteally application. The background features a sunset over a forest with a deer silhouette. A red error message box at the top left contains the text "Incorrect Login or Password". The main form has fields for "Login" and "Password", each with a placeholder icon (user and lock). Below the fields is a "Remember Me!" checkbox. A large pink "Login" button is centered at the bottom. At the very bottom, there's a link "Don't have an account? Register Me!".

Przykład widoku z walidacją `createNote.html`:

The screenshot shows the "CreateNote" form from the Noteally application. The title bar says "CreateNote" with a close button. The form has three fields: "Title" (with placeholder "Title"), "Content" (with placeholder "Content"), and "Link" (with placeholder "Link"). Two red error message boxes point to the "Title" and "Content" fields. The "Title" box says "Size must be between 3 and 20" and "Title is required". The "Content" box says "Size must be between 5 and 500" and "Content is required". A pink "Create Note" button is at the bottom.

Przykład widoku z walidacją `createCatalog.html`:



Sesja

W projekcie używamy sesji do pobierania informacji o odpowiednim użytkowniku. Listener **AuthenticationApplicationListener.java** przy udanej autentykacji dodaje do sesji atrybut **userId**. Atrybut ten jest wykorzystywany między innymi w widoku **catalogs.html** do pobrania katalogów tego użytkownika, który jest aktualnie zalogowany. Ten atrybut sesji służy nam też do sprawdzania, który użytkownik jest zalogowany i czy ma on dostęp do konkretnych funkcjonalności.

AuthenticationApplicationListener.java:

```
1 package pl.noteally.config;
2
3
4 import jakarta.servlet.http.HttpSession;
5 import lombok.AllArgsConstructor;
6 import org.springframework.context.event.EventListener;
7 import org.springframework.security.authentication.event.InteractiveAuthenticationSuccessEvent;
8 import org.springframework.stereotype.Component;
9 import pl.noteally.data.User;
10
11 @Component
12 @AllArgsConstructor
13 public class AuthenticationApplicationListener {
14
15     private final HttpSession session;
16     @EventListener
17     @HandlesEvent(InteractiveAuthenticationSuccessEvent.class)
18     public void handleInteractiveAuthenticationSuccess(InteractiveAuthenticationSuccessEvent event) {
19         User user = (User) event.getAuthentication().getPrincipal();
20         session.setAttribute("userId", user.getId());
21     }
}
```

Fragment kontrolera **CatalogsController.java** pobierający katalogi na podstawie atrybutu sesji **userId**:

```
31 @GetMapping("")
32 public String getCatalogsByUserId(Model model, HttpSession session, HttpServletRequest request) {
33     Optional<String> sortValue = Arrays.stream(request.getCookies()).filter(
34         c -> c.getName().equals("catalogCookie" + session.getAttribute("userId"))).map(Cookie::getValue).findAny();
35 
36     if(sortValue.isEmpty())
37         return "redirect:/login";
38 
39     if (sortValue.get().equals("ASC"))
40         return "redirect:/catalogs/ASC";
41     else if (sortValue.get().equals("DESC"))
42         return "redirect:/catalogs/DESC";
43     else if (sortValue.get().equals("notesASC"))
44         return "redirect:/catalogs/notesASC";
45     else if (sortValue.get().equals("notesDESC"))
46         return "redirect:/catalogs/notesDESC";
47 
48     List<Catalog> catalogList = catalogService.getCatalogsByUserId((Integer) session.getAttribute("userId"));
49     Optional<User> user = userService.getUserById((Integer) session.getAttribute("userId"));
50     model.addAttribute("catalogs", catalogList);
51     model.addAttribute("user", user.get());
52     return "catalogs";
53 }
```

Ciasteczka

Ciasteczka służące do zapamiętywania kryteriów i kierunków sortowania tworzone są przy rejestracji użytkownika.

Kontroler **Log_Reg_Controller**, fragment metody **register()**:

```
82     Cookie catalogCookie = new Cookie( name: "catalogCookie" + user.getId(), value: "default");
83     catalogCookie.setMaxAge(60 * 60 * 24 * 365 * 10);
84     Cookie noteCookie = new Cookie( name: "noteCookie" + user.getId(), value: "default");
85     noteCookie.setMaxAge(60 * 60 * 24 * 365 * 10);
86     response.addCookie(catalogCookie);
87     response.addCookie(noteCookie);
88 }
```

Usługa REST

Na potrzeby serwisu Noteally, w oddzielnym projekcie **IsNamePolishREST** została skonfigurowana usługa REST pod adresem **localhost:9090**. Usługa ta jest wykorzystywana przy rejestracji użytkownika i porównuje ona wpisane imię do bazy polskich imion **polishNames.txt**.

Fragment serwisu **IsNamePolishService.java** znajdującego się w projekcie **IsNamePolishREST** jest on odpowiedzialny za porównanie odebranego imienia z plikiem **polishNames.txt**:

```
12  @Service
13  public class IsNamePolishService {
14
15      private List<String> nameList;
16      public IsNamePolishService()
17      {
18          nameList = new ArrayList<>();
19          try{
20              BufferedReader reader = new BufferedReader(new FileReader(new File( pathname: "src/polishNames.txt")));
21              String name;
22              while ((name = reader.readLine()) != null)
23              {
24                  nameList.add(name);
25              }
26          } catch(IOException e) {
27              e.printStackTrace();
28          }
29      }
30      public boolean isNamePolish(String name)
31      {
32          if(name == null) return false;
33          if(nameList.contains(name)) return true;
34          else return false;
35      }
36  }
```

Fragment serwisu **RestNameService** odpowiedzialnego za wysyłanie zapytań do naszego serwera REST:

```
7  @Service
8  public class RestNameService {
9
10     @Autowired
11     private RestTemplate restTemplate;
12
13     public boolean isNamePolish(String name)
14     {
15         Boolean response = restTemplate.getForObject( url: "http://localhost:9090/" + name, Boolean.class);
16         if(response == null) response = false;
17         return response;
18     }
19 }
```

Fragment kontrolera **Log_Reg_Controller** (metoda **register**) odpowiedzialny za sprawdzenie, czy podane imię jest polskie:

```
66             if(!restNameService.isNamePolish(user.getName())) {
67                 model.addAttribute( attributeName: "polishNameError", attributeValue: "Name must be Polish");
68                 errors = true;
69             }
```

Wygląd - (WCAG 2.1)

Do sprawdzenia poziomu dostępności naszej aplikacji użyto wtyczki do przeglądarki WAVE. Pokazuje ona tylko kilka nieznacznych błędów. Dotyczą one braku tekstu przy linkach służących do sortowania lub tworzenia zasobów. Wyświetlają się one, gdyż postanowiliśmy zastosować ikonki, zamiast tekstu.

