

<b>Wydział Informatyki Politechniki Białostockiej</b> Katedra Mediów Cyfrowych i Grafiki Komputerowej <b>Pracownia</b> Programowanie aplikacji WWW w Javie	<b>Data przeprowadzenia projektu:</b> maj/czerwiec 2023 <b>Data oddania projektu:</b> 13.06.2023
<b>Sprawozdanie z projektu</b> Temat: Aplikacja zapisująca notatki. <b>Grupa realizująca projekt:</b> 1) Bartosz Wiśniewski 2) Maciej Waluk 3) Paweł Picewicz 4) Adrian Toczydłowski	<b>Prowadzący:</b> dr inż. Urszula Kuźlewska

## 1) Cel projektu

Celem projektu było utworzenie aplikacji do zapisywania i udostępniania wartych uwagi informacji, np. strony www, listy zakupów, zaproszenia na spotkanie.

## 2) Realizacja projektu

### Notatnik

Aplikacja ma służyć do zapisywania i udostępniania wartych uwagi informacji, np. strony www, listy zakupów, zaproszenia na spotkanie, notatki. Informacja ma składać się z: tytułu (3-20), treści (5-500), [linku], daty dodania (aktualna, format dd-mm-yyyy), [daty przypomnienia], kategorii (3-20, małe litery). Każdy użytkownik ma posiadać: imię (3-20, litery, pierwsza duża), nazwisko (3-50, litery, pierwsza duża), login (3-20, małe litery), hasło (co najmniej 5 znaków), wiek (min. 18 lat). Główną funkcjonalnością jest umożliwienie zapisania na swoim koncie ciekawych informacji, udostępniania ich innym oraz wygodnego przeglądania i przeszukiwania w dogodnym czasie.

### Funkcjonalności FULL\_USER

- Dodanie/edycja/usunięcie przez siebie zebranych informacji – 5p

Wymagania projektowe dotyczące dodawania, edytowania i usuwania notatek spełniają metody zawarte w kontrolerze NoteController. Metoda addNote pozwala użytkownikowi na dodanie nowej notatki, walidując dane wejściowe i zapisując notatkę do bazy danych. Metoda editNoteForm umożliwia użytkownikowi edycję istniejącej notatki, pobierając jej szczegóły i umożliwiając ich modyfikację na formularzu. Metoda editNote aktualizuje notatkę o określonym identyfikatorze na podstawie wprowadzonych zmian. Metoda deleteNote usuwa notatkę o określonym identyfikatorze. Każda z tych metod przekierowuje użytkownika na odpowiednią stronę po zakończeniu operacji.

## Dodawanie

```
no usages
@PostMapping("/add")
public String addNote(
    @ModelAttribute("note") @Valid Note note,
    BindingResult bindingResult,
    Model model,
    Principal principal
) {
    if (bindingResult.hasErrors()){
        System.out.println(bindingResult.getAllErrors());
        return "note-form";
    }
    else {
        try {
            noteService.save(note, principal.getName());
        } catch (InvalidCategoryException exception) {
            model.addAttribute("errorMessage", exception.getMessage());
            return "note-form";
        }
        return "redirect:/notes/all";
    }
}
```

## Edytowanie

```
no usages
@GetMapping("/{id}/edit")
public String editNoteForm(@PathVariable("id") Long id, Model model, Principal principal) {
    Note note = noteService.findById(id);
    model.addAttribute("note", note);
    model.addAttribute("categories", noteService.getAllCategories(principal.getName()));
    return "edit-note-form";
}

no usages
@PostMapping("/{id}/edit")
public String editNote(
    @PathVariable("id") Long id,
    @ModelAttribute("note") @Valid Note updatedNote,
    BindingResult bindingResult
) {
    if (bindingResult.hasErrors()){
        System.out.println(bindingResult.getAllErrors());
        return "edit-note-form";
    }
    else {
        noteService.editNote(id, updatedNote);
        return "redirect:/notes/all";
    }
}
```

```
no usages
@GetMapping("/{id}/delete")
public String deleteNote(@PathVariable("id") Long id) {
    noteService.deleteNoteById(id);
    return "redirect:/notes/all";
}
```

- Walidacja formularza – 1p

Pozwala na to użycie adnotacji walidacyjnych tj. @NotBlank, @Size i @Pattern w klasie Note, przez co dane wprowadzane w formularzach są walidowane zgodnie z określonymi regułami. Umożliwia to sprawdzenie poprawności danych przed ich zapisaniem w bazie danych.

```
public class Note {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @Column(name = "Tytuł")
    @NotBlank(message = "Tytuł nie może być pusty")
    @Size(min = 3, max = 20, message = "Tytuł musi mieć od 3 do 20 znaków.")
    private String tytuł;
    no usages
    @Column(name = "Treść")
    @NotBlank(message = "Treść notatki nie może być pusta")
    @Size(min = 5, max = 500, message = "Treść notatki musi mieć od 5 do 500 znaków.")
    private String tresc;
    no usages
    @Column(name = "Dodano")
    private LocalDate dataDodania;
    no usages
    @Column(name = "Przypomnienie")
    private LocalDate dataPrzypomnienia;
    no usages
    @Column(name = "Kategoria")
    @NotBlank(message = "Kategoria nie może być pusta")
    @Size(min = 3, max = 20, message = "Kategoria musi zawierać od 3 do 20 znaków.")
    @Pattern(regexp = "^[a-z]+$", message = "W nazwie kategorii mogą znajdować się tylko małe litery.")
    private String kategoria;
    no usages
    @Column(name = "Link")
    private String link;
}
```

- Edycja na danych bieżących – 1p

Pozwalają na to metody editNoteForm i editNote znajdujące się w kontrolerze NoteController, które umożliwiają użytkownikowi edycję istniejącej notatki na podstawie identyfikatora notatki. W obu przypadkach operujemy na istniejących danych notatki i modyfikujemy je na podstawie wprowadzonych zmian przez użytkownika, a co za tym idzie, operują one na danych bieżących.

- Dodanie nowej kategorii – **1p**

Zapewnia to kontroler `NoteController` i jego metody `addNoteForm` i `addNote`, które umożliwiają użytkownikowi dodawanie nowej notatki wraz z możliwością wprowadzania nowych kategorii. Obsługują one różne przypadki, takie jak walidacja danych czy obsługa istniejących kategorii.

- Wyświetlenie udostępnionych przez innych informacji – **2p**

Odpowiedzialna jest za to metoda `sharedNotes` z kontrolera `NoteController` – pobiera ona i filtruje notatki, które są udostępnione przez innych użytkowników, a następnie przekazuje je do widoku w celu wyświetlenia.

```
no usages
@GetMapping("/shared")
public String sharedNotes(Model model, Principal principal) {
    List<Note> sharedNotes = noteService.findAll(principal.getName())
        .stream()
        .filter(Note::isUdostepniona)
        .collect(Collectors.toList());
    model.addAttribute("notes", sharedNotes);
    return "shared-notes";
}
```

- Udostępnienie w linku – **0.5p**

Zapewnia to metoda `getNoteByLink` z kontrolera `NoteController`, która umożliwia uzyskanie notatki na podstawie linku i przekazuje ją do widoku w celu wyświetlenia szczegółów. Dzięki temu użytkownik może uzyskać dostęp do konkretnej notatki poprzez link, który został udostępniony.

```
no usages
@GetMapping("/all/{link}")
public String getNoteByLink(@PathVariable("link") String link, Model model) {
    Note note = noteService.findByLink(link);
    model.addAttribute("note", note);
    return "note-details";
}
```

- Udostępnienie ze wskazaniem na konkretnego użytkownika – **0.5p**

Pozwala na to NoteController i jego metody shareNoteForm i shareNote, które umożliwiają udostępnienie notatki konkretnemu użytkownikowi poprzez pobranie listy użytkowników, wyświetlenie formularza wyboru użytkownika i obsługę przekazywanych danych, tak aby utworzyć i zapisać udostępnioną notatkę.

```
@GetMapping("/{id}/share")
public String shareNoteForm(@PathVariable("id") Long id, Model model) {
    List<User> users = userService.getAllUsers();
    Note note = noteService.findById(id);
    model.addAttribute("note", note);
    model.addAttribute("users", users);
    return "share-note-form";
}

no usages
@PostMapping("/{id}/share")
public String shareNote(
    @PathVariable("id") Long noteID,
    @RequestParam("userId") Long userID
)
{
    Note sharedNote = new Note();
    Note note = noteService.findById(noteID);

    sharedNote.setUser(userService.findById(userID));
    sharedNote.setUdostepniona(true);
    sharedNote.setTresc(note.getTresc());
    sharedNote.setKategoria(note.getKategoria());
    sharedNote.setDataDodania(note.getDataDodania());
    sharedNote.setTytul(note.getTytul());
    sharedNote.setLink(note.getLink());
    sharedNote.setDataPrzypomnienia(note.getDataPrzypomnienia());

    noteService.save(sharedNote, sharedNote.getUser().getUsername());

    return "redirect:/notes/all";
}
```

- sortowanie w obu kierunkach, zapamiętanie kierunków i kryteriów sortowania oraz filtrowanie według daty (od aktualnej) i kategorii (od najbardziej popularnej) – **5p**

Wszystkie powyższe funkcjonalności zapewnia metoda getAll w kontrolerze NoteController, która umożliwia sortowanie i filtrowanie notatek. Parametr sortOption służy tutaj do wyboru opcji sortowania notatek, a categoryFilter służy do filtrowania notatek według kategorii. Dodatkowo, metoda obsługuje parametry startDate i endDate do filtrowania notatek według zakresu dat oraz zapamiętuje wybrane opcje sortowania i filtry za pomocą ciasteczek, dzięki czemu użytkownik może zachować preferencje przy kolejnych żądaniach.

- logowanie – 1p

Pozwalają na to następujące klasy: klasa LoginController obsługuje ścieżkę "/login" i umożliwia wyświetlenie formularza logowania dla użytkowników, którzy wchodzą na tę stronę oraz klasa SpringSecurity, która konfiguruje zabezpieczenia aplikacji. Metoda filterChain() definiuje konfigurację zabezpieczeń dla różnych ścieżek. Ścieżka "/login" jest ustawiona jako dostępna dla wszystkich, co oznacza, że wszyscy użytkownicy mają do niej dostęp. Metoda formLogin() definiuje stronę logowania, adres przetwarzania logowania oraz stronę powodzenia logowania. Dodatkowo, klasa SpringSecurity używa DaoAuthenticationProvider do uwierzytelniania użytkowników na podstawie informacji z bazy danych. Dostawca ten jest skonfigurowany z użyciem dostawcy usługi użytkowników userDetailsService i kodera hasła passwordEncoder().

### LoginController

```
no usages
@Controller
@RequestMapping("/login")
@AllArgsConstructor
public class LoginController {
    no usages
    @GetMapping
    public String showLogin() {
        return "login-form";
    }
}
```

### SpringSecurity

```
no usages
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests((authorize) -> {
            authorize.requestMatchers(...patterns: "/welcome", "/register/**").permitAll()
                .requestMatchers(...patterns: "/notes/all", "notes/all/*", "/notes/shared", "/notes/reminded-notes")
                .requestMatchers(...patterns: "/notes", "notes/add", "notes/*/edit", "notes/*/delete", "notes/sh
                .requestMatchers(...patterns: "/admin", "/admin/**").hasRole("ADMIN")
        })
        .formLogin(
            form -> form
                .loginPage("/login")
                .loginProcessingUrl("/login")
                .defaultSuccessUrl("/notes/reminded-notes")
                .permitAll()
        )
        .logout(
            logout -> logout
                .logoutRequestMatcher(new AntPathRequestMatcher(pattern: "/logout"))
                .permitAll()
        )
        .headers(headers -> headers.frameOptions().disable());
    return http.build();
}
```

## Funkcjonalności NIEZALOGOWANY

- Rejestracja – 1p

Klasa `RegistrationController` obsługuje rejestrację użytkowników poprzez udostępnienie formularza rejestracji i przetworzenie danych przesłanych z tego formularza. Metoda `registrationForm()` obsługuje żądanie GET dla ścieżki `"/register"` i zwraca widok formularza rejestracji. Metoda `registrationProcess()` obsługuje żądanie POST dla tej samej ścieżki i przetwarza dane rejestracyjne. Jeśli dane są poprawne, użytkownik zostaje zapisany, a następnie przekierowany na stronę logowania. Klasa `SpringSecurity` konfiguruje zabezpieczenia aplikacji. W tym przypadku, rejestracja jest ustawiona jako dostępna dla wszystkich użytkowników, aby każdy mógł mieć możliwość utworzenia konta. Konfiguracja ta jest realizowana poprzez metodę `filterChain()`.

```
no usages
@Controller
@AllArgsConstructor
@RequestMapping("/register")
public class RegistrationController {
    1 usage
    private UserService userService;
    no usages
    @GetMapping("")
    public String registrationForm(Model model) {
        model.addAttribute(attributeName: "user", new User());
        return "registration-form";
    }
    no usages
    @PostMapping("")
    public String registrationProcess(
        @ModelAttribute("user") @Valid User user,
        BindingResult bindingResult
    ) {
        if (bindingResult.hasErrors()){
            System.out.println(bindingResult.getAllErrors());
            return "registration-form";
        }
        else {
            userService.save(user);
            return "redirect:/login";
        }
    }
}
```

- Walidacja formularza – 1p

Pozwala na to użycie adnotacji walidacyjnych tj. @NotBlank, @Size i @Pattern w klasie User, przez co dane wprowadzane w formularzach są walidowane zgodnie z określonymi regułami. Umożliwia to sprawdzenie poprawności danych przed ich zapisaniem w bazie danych.

```
@AllArgsConstructor
@Entity
@Table(name = "Uzytkownicy")
public class User implements UserDetails {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    no usages
    @Column(name = "Imie")
    @NotBlank(message = "Imię jest wymagane")
    @Size(min = 3, max = 20, message = "Imię powinno mieć od 3 do 20 znaków")
    @Pattern(regexp = "^[A-Z][a-zA-Z]*$", message = "Imię powinno składać się z liter")
    private String firstName;

    no usages
    @Column(name = "Nazwisko")
    @NotBlank(message = "Nazwisko jest wymagane")
    @Size(min = 3, max = 50, message = "Nazwisko powinno mieć od 3 do 50 znaków")
    @Pattern(regexp = "^[A-Z][a-zA-Z]*$", message = "Nazwisko powinno składać się z liter")
    private String lastName;

    no usages
    @Column(name = "Login", unique = true)
    @NotBlank(message = "Login jest wymagany")
    @Size(min = 3, max = 20, message = "Login powinien mieć od 3 do 20 znaków")
    @Pattern(regexp = "^[a-z]+$", message = "Login powinien składać się z małych liter")
    private String username;

    no usages
    @Column(name = "Haslo")
```

- Strona powitalna – 1p

To wymaganie dotyczące strony powitalnej jest spełnione przez WelcomeController. Użytkownik, który odwiedza ścieżkę "/welcome", otrzymuje odpowiedni widok "welcome-page". Jest to pierwsza strona, którą widzi po bezpośrednim wejściu na stronę główną aplikacji.

```
no usages
@Controller
public class WelcomeController {

    no usages
    @GetMapping("/welcome")
    public String welcome() { return "welcome-page"; }
}
```



- Wyświetlenie informacji z udostępnionego linku – **1p**

Funkcjonalność wyświetlania informacji z udostępnionego linku jest realizowana w metodzie `sharedNotes` w kontrolerze `NoteController` poprzez filtrowanie i zbieranie udostępnionych notatek dla zalogowanego użytkownika. Lista przefiltrowanych notatek jest przekazywana do widoku poprzez obiekt `Model`, a następnie wyświetlana w widoku `"shared-notes"`.

```
no usages
@GetMapping("/shared")
public String sharedNotes(Model model, Principal principal) {
    List<Note> sharedNotes = noteService.findAll(principal.getName())
        .stream()
        .filter(Note::isUdostepniona)
        .collect(Collectors.toList());
    model.addAttribute("notes", sharedNotes);
    return "shared-notes";
}
```

## Funkcjonalności ADMIN

- Wyświetlenie listy użytkowników – **1p**

Metoda `getUsers` z kontrolera `AdminController` spełnia funkcjonalność wyświetlania listy użytkowników, ponieważ w odpowiedzi na żądanie GET do odpowiedniego endpointa, przekazuje on listę użytkowników do modelu i wyświetla widok `"admin-view"`, prezentujący tę listę użytkowników.

```
@GetMapping("/users")
public String getUsers(Model model) {
    model.addAttribute("users", userService.findAll());
    return "admin-view";
}
```

- Zarządzanie rolami – **1p**

Metody `showEdit` i `editUser` w kontrolerze `AdminController` umożliwiają administratorowi wyświetlać formularz edycji informacji o użytkowniku, w tym zmieniać rolę, oraz zapisywać te zmiany w bazie danych. Pozwala to na efektywne zarządzanie rolami użytkowników w systemie.

no usages

```
@GetMapping("/{id}/edit")
public String showEdit(@PathVariable("id") Long id, Model model) {
    User user = userService.findById(id);
    model.addAttribute(attributeName: "user", user);
    return "edit-user";
}
```

no usages

```
@PostMapping("/{id}/edit")
public String editUser(
    @PathVariable("id") Long id,
    @ModelAttribute("user") @Valid User updatedUser
) {
    userService.editUser(id, updatedUser);
    return "redirect:/admin/users";
}
```

no usages

```
@GetMapping("/{id}/edit")
public String showEdit(@PathVariable("id") Long id, Model model) {
    User user = userService.findById(id);
    model.addAttribute(attributeName: "user", user);
    return "edit-user";
}
```

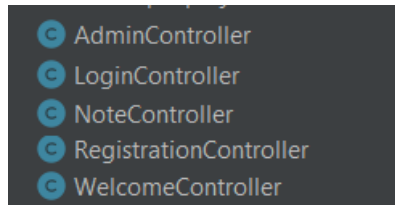
no usages

```
@PostMapping("/{id}/edit")
public String editUser(
    @PathVariable("id") Long id,
    @ModelAttribute("user") @Valid User updatedUser
) {
    userService.editUser(id, updatedUser);
    return "redirect:/admin/users";
}
```

## Elementy techniczne

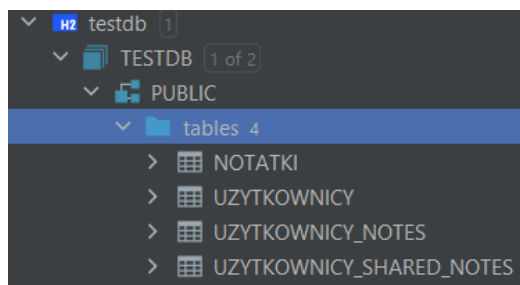
- Kontrolery – **2p**

W projekcie zostało użyte pięć kontrolerów: AdminController – do operacji używanych przez administratorów, LoginController – do obsługi logowania, NoteController – do obsługi operacji na notatkach, RegistrationController – do obsługi rejestracji i WelcomeController – do obsługi żądania strony powitalnej.



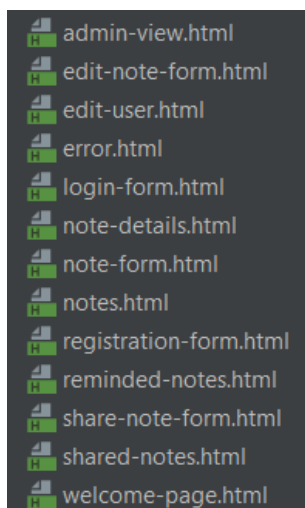
- Baza danych (co najmniej 2 tabele z relacją) – **5p**

Baza danych H2 w projekcie zawiera 4 tabele: Notatki – zawierająca wszystkie notatki w systemie, Użytkownicy – zawierająca wszystkich użytkowników w systemie oraz Użytkownicy\_Notes i Użytkownicy\_Shared\_Notes, które są dwiema tabelami łączącymi.



- Widoki: formularze z walidacją i 5 różnych znaczników Thymeleafa – **5p**

Utworzono trzynaście widoków nowych widoków, gdzie każdy z nich zawiera jakieś ze znaczników Thymeleafa. Przykłady użytych znaczników to między innymi: text, field, if, errors, object i selected. Formularze edit-note-form oraz note-form mają zapewnioną walidację notatek, aby ich treść była zgodna z wytycznymi projektu.



- Sesja i Spring Security – 7p

W projekcie sesja jest używana w kontekście zarządzania uwierzytelnianiem i autoryzacją użytkowników w aplikacji. Sesja jest wykorzystywana do przechowywania informacji o zalogowanym użytkowniku oraz do śledzenia jego stanu podczas interakcji z aplikacją. Dzieje się to w klasie SpringSecurity, gdzie po pomyślnym uwierzytelnieniu użytkownika, informacje o sesji są przechowywane przez Spring Security, dzięki czemu użytkownik nie musi ponownie wpisywać swoich danych uwierzytelniających przy kolejnych żądaniach do chronionych zasobów. Dodatkowo, przypisane role użytkownika są także przechowywane w sesji, co umożliwia Spring Security sprawdzanie, czy użytkownik posiada odpowiednie uprawnienia do dostępu do określonych zasobów. Po wykonaniu operacji wylogowania, sesja użytkownika jest usuwana, co powoduje usunięcie informacji o jego zalogowaniu i uprawnieniach.

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests((authorize) -> {
            authorize.requestMatchers(...patterns: "/welcome", "/register/**").permitAll()
                .requestMatchers(...patterns: "/notes/all", "notes/all/*", "/notes/shared", "/notes/reminded-notes")
                .requestMatchers(...patterns: "/notes", "notes/add", "notes/*/edit", "notes/*/delete", "notes/sh
                .requestMatchers(...patterns: "/admin", "/admin/**").hasRole("ADMIN")
        })
        .formLogin(
            form -> form
                .loginPage("/login")
                .loginProcessingUrl("/login")
                .defaultSuccessUrl("/notes/reminded-notes")
                .permitAll()
        )
        .logout(
            logout -> logout
                .logoutRequestMatcher(new AntPathRequestMatcher(pattern: "/logout"))
                .permitAll()
        )
        .headers(headers -> headers.frameOptions().disable());
    return http.build();
}

no usages
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setPasswordEncoder(passwordEncoder());
    authProvider.setUserDetailsService(userDetailsService);
    return authProvider;
}
```

- Ciasteczka – 1p

Ciasteczka są wykorzystywane w metodzie getAll w kontrolerze NoteController w celu zapamiętania preferencji sortowania i filtrowania notatek wybranych przez użytkownika. Dzięki temu, gdy użytkownik odświeża stronę lub przechodzi do innej części aplikacji, wcześniej wybrane opcje sortowania i filtrowania są zachowane i stosowane w kolejnych żądaniach.

```

@GetMapping("/all")
public String getAll(
    @RequestParam(value = "sortOption", required = false) String sortOption,
    @RequestParam(value = "categoryFilter", required = false) String categoryFilter,
    @RequestParam(value = "startDate", required = false) String startDate,
    @RequestParam(value = "endDate", required = false) String endDate,
    Model model,
    HttpServletResponse response,
    HttpServletRequest request,
    Principal principal
) {
    List<Note> notes;

    if (sortOption == null) {
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals("sortOption")) {
                    sortOption = cookie.getValue();
                    break;
                }
            }
        }
    } else {
        Cookie sortOptionCookie = new Cookie("sortOption", sortOption);
        sortOptionCookie.setMaxAge(86400);
        response.addCookie(sortOptionCookie);
    }

    if (sortOption != null && !sortOption.isEmpty()) {
        if (sortOption.equals("Popularnosc")) {

```

```

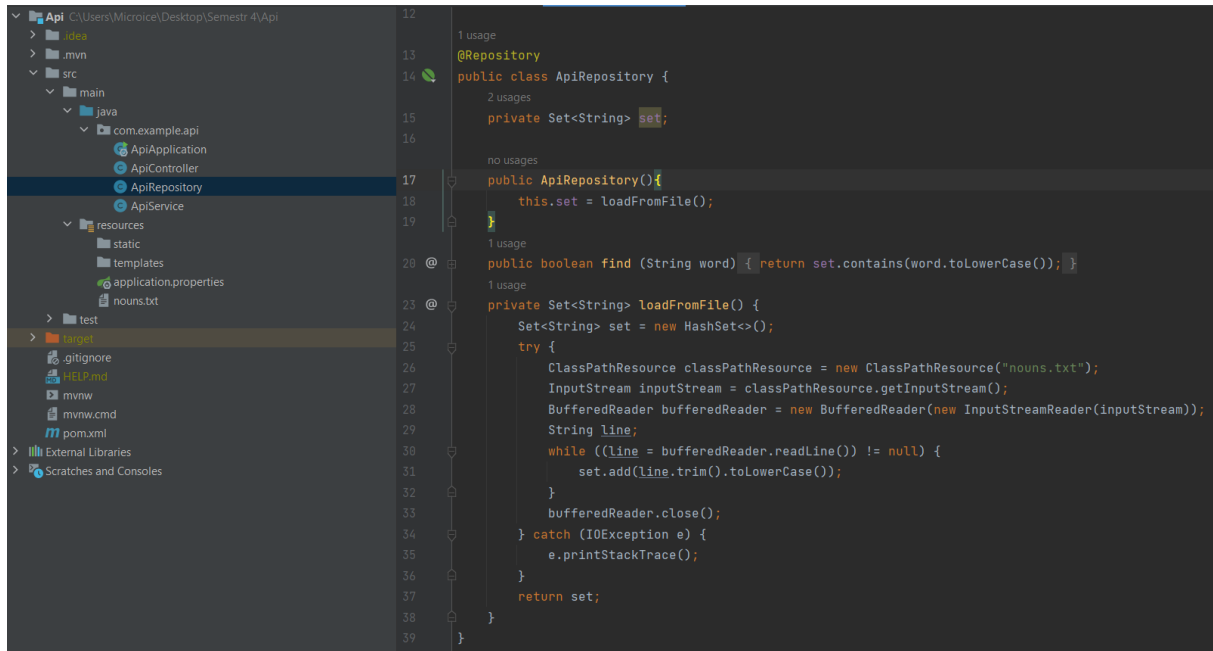
            if (sortOption.equals("Popularnosc")) {
                notes = noteService.findAllSortedBy(principal.getName(), sortOption: "Popularnosc");
            } else {
                notes = noteService.findAllSortedBy(principal.getName(), sortOption);
            }
        } else {
            notes = noteService.findAll(principal.getName());
        }

    if (categoryFilter == null) {
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if (cookie.getName().equals("categoryFilter")) {
                    categoryFilter = cookie.getValue();
                    break;
                }
            }
        }
    } else {
        Cookie categoryFilterCookie = new Cookie("categoryFilter", categoryFilter);
        categoryFilterCookie.setMaxAge(86400);
        response.addCookie(categoryFilterCookie);
    }

```

- Usługa REST (do weryfikacji nazwy kategorii w słowniku) i Klient REST – 5p

W tym celu napisano własne REST API, którego zadaniem jest sprawdzenie czy podana kategoria jest rzeczownikiem znajdującym się w naszym pliku.

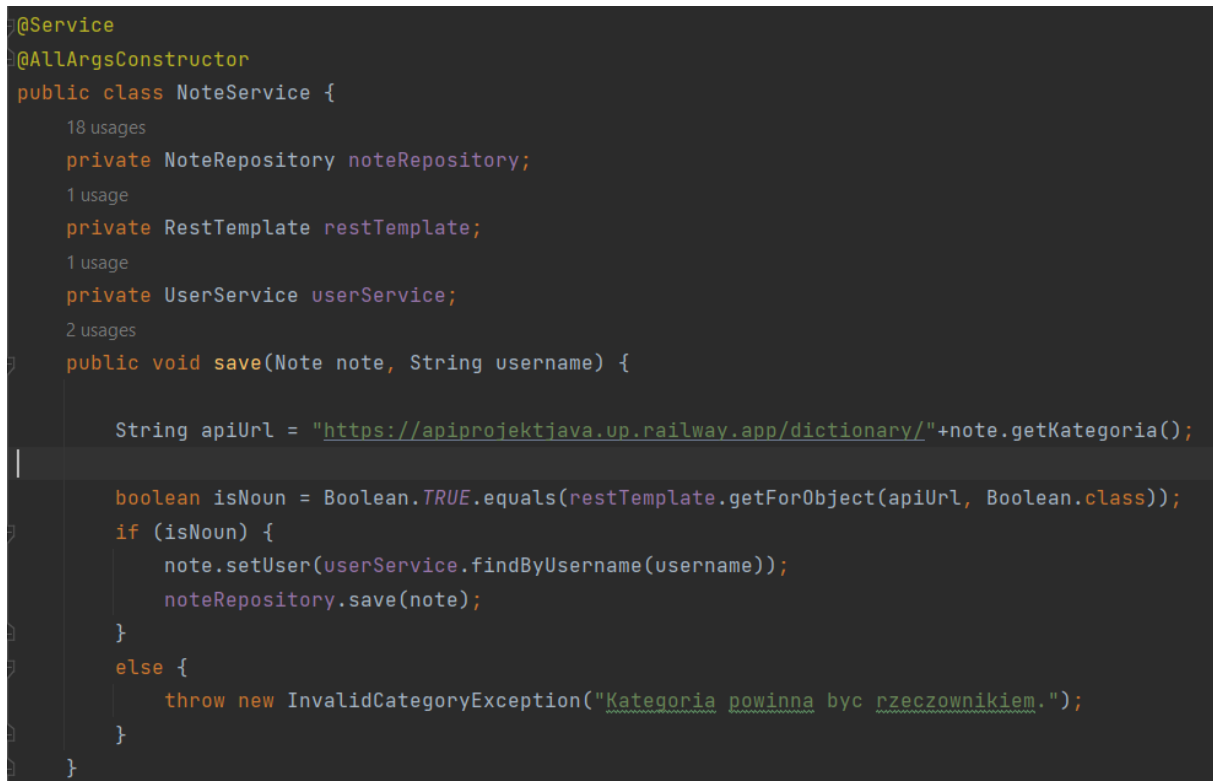


```

12
13
14 @Repository
15 public class ApiRepository {
16     private Set<String> set;
17
18     public ApiRepository() {
19         this.set = loadFromFile();
20     }
21
22     public boolean find (String word) { return set.contains(word.toLowerCase()); }
23
24     private Set<String> loadFromFile() {
25         Set<String> set = new HashSet<>();
26         try {
27             ClassPathResource classPathResource = new ClassPathResource("nouns.txt");
28             InputStream inputStream = classPathResource.getInputStream();
29             BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
30             String line;
31             while ((line = bufferedReader.readLine()) != null) {
32                 set.add(line.trim().toLowerCase());
33             }
34             bufferedReader.close();
35         } catch (IOException e) {
36             e.printStackTrace();
37         }
38         return set;
39     }

```

Następnie korzystając z utworzonego REST API postawionego na hostingu railway.app, używamy tej funkcjonalności w naszym projekcie w serwisie NoteService.



```

@Service
@AllArgsConstructor
public class NoteService {
    18 usages
    private NoteRepository noteRepository;
    1 usage
    private RestTemplate restTemplate;
    1 usage
    private UserService userService;
    2 usages
    public void save(Note note, String username) {
        String apiUrl = "https://apiprojektjava.up.railway.app/dictionary/"+note.getKategoria();

        boolean isNoun = Boolean.TRUE.equals(restTemplate.getForObject(apiUrl, Boolean.class));
        if (isNoun) {
            note.setUser(userService.findByUsername(username));
            noteRepository.save(note);
        }
        else {
            throw new InvalidCategoryException("Kategoria powinna byc rzeczownikiem.");
        }
    }
}

```

- Informacje z datą przypomnienia zgodną z datą bieżącą powinny być wyświetlane w specjalnym widoku, który pojawia się zaraz po zalogowaniu - **dodatkowe**

W tym celu metoda remindedNote z kontrolera NoteController pobiera wszystkie notatki dla zalogowanego użytkownika. Następnie filtruje notatki, wybierając tylko te, których data przypomnienia jest zgodna z bieżącą datą. Jeśli istnieją takie notatki, są one przekazywane do widoku "reminded-notes". W przeciwnym przypadku użytkownik jest przekierowywany do widoku listy wszystkich notatek.

```
no usages
@GetMapping("/reminded-notes")
public String remindedNotes(Model model, Principal principal) {
    LocalDate now = LocalDate.now();
    List<Note> remindedNotes = noteService.findAll(principal.getName())
        .stream()
        .filter(note -> now.equals(note.getDataPrzypomnienia()))
        .collect(Collectors.toList());
    if (remindedNotes.isEmpty()){
        return "redirect:/notes/all";
    }
    model.addAttribute("notes", remindedNotes);
    return "reminded-notes";
}
```

## Standardy WCAG na użytych stronach

/notes/all

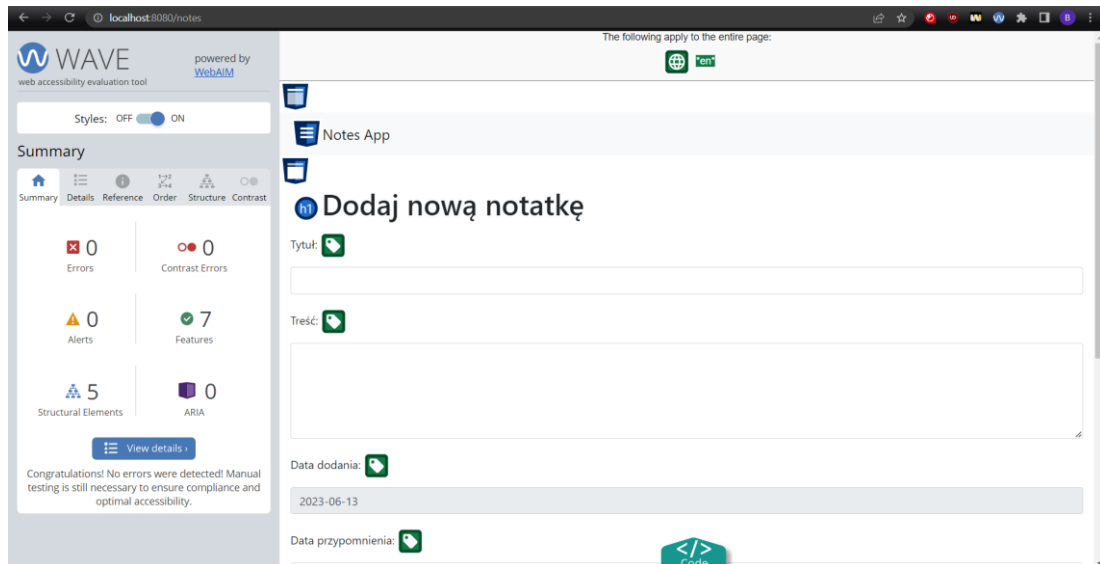
The screenshot shows a web application interface for managing notes. On the left, there is a sidebar with a 'Summary' section from the WebAIM accessibility tool. The summary indicates 0 errors, 0 contrast errors, 0 alerts, 12 features, 11 structural elements, and 0 ARIA issues. A message at the bottom of the sidebar states: 'Congratulations! No errors were detected! Manual testing is still necessary to ensure compliance and optimal accessibility.'

The main content area is titled 'Notatki' (Notes) and includes a 'Note List' header. It features a table with columns: Tytuł (Title), Data dodania (Date added), Kategoria (Category), Treść (Content), and Akcje (Actions). The table contains two entries:

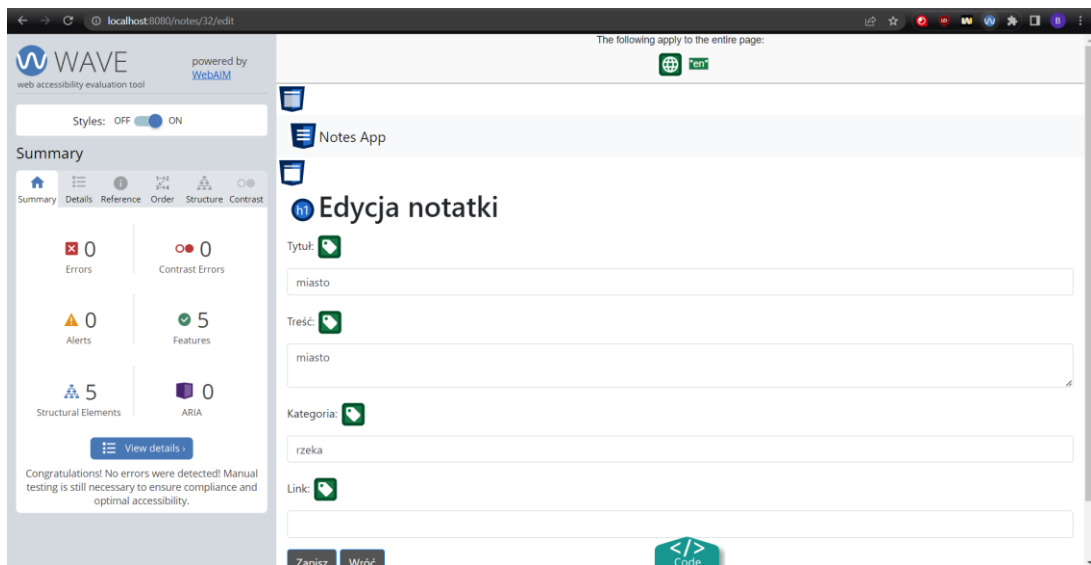
Tytuł	Data dodania	Kategoria	Treść	Akcje
miasto	2023-06-08	rzeka	miasto	Edytuj Udostępnij Usuń
druga notatka	2023-06-09	dom	notatka	Edytuj Udostępnij Usuń

At the top of the notes list, there are filters for sorting (Sortowanie: Data dodania (rosnąco)), filtering (Filtrowanie: Wszystkie), and date format (Początek: dd.mm.rrrr, Koniec: dd.mm.rrrr). A 'Zatwierdź' (Confirm) button is also present.

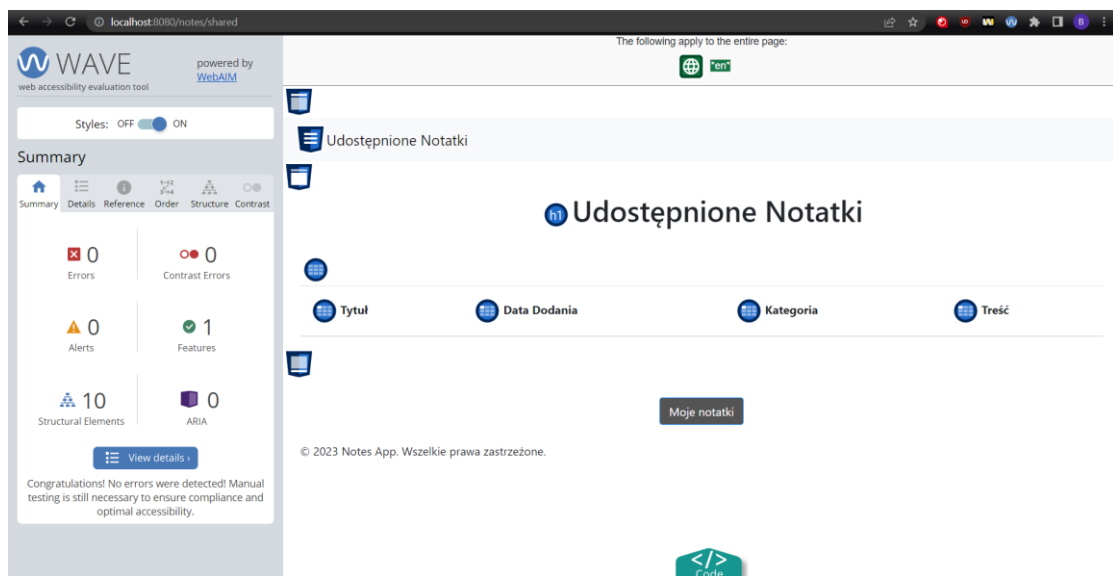
## /notes



## /notes/{id}/edit



## /notes/shared





## /admin/users

The following apply to the entire page:

Users List

### Użytkownicy

Imię	Nazwisko	Login	Hasło	Wiek	Rola	Akcje
Bartosz	Wisniewski	tygrys	\$2a\$10\$3x.ID.H0RO.UC.7YVCeM1.b2.ANjKZauq1GE9.rRt/BcGxZPq84L	25	ROLE_ADMIN	Zmień Uprawnie
Lionel	Messi	messi	\$2a\$10\$JpEcFkqUIrtVDen6GP85.UlnobdlxX1iApVyLrCKhMEKz2BH7KW	35	ROLE_FULL_USER	Zmień Uprawnie
Adam	Malysz	malysz	\$2a\$10\$1z9Qy.NHzlDlzmHgnBG4eGmkJCyc9Cai3415RFsEsm/XmRAJBji6	52	ROLE_LIMITED_USER	Zmień Uprawnie
Tomek	Tomek	tomek	\$2a\$10\$TLTaqwO/H3ioLhnl/OuOoiB8P9CmBXI7mEzfy7PibgVfWG	21	ROLE_FULL_USER	Zmień Uprawnie

## /login

The following apply to the entire page:

Notes App

### Logowanie do aplikacji

Nazwa użytkownika:

Hasło:

Zaloguj się

© 2023 Notes App. Wszelkie prawa zastrzeżone.

## /register

The following apply to the entire page:

Notes App

### Rejestracja w aplikacji do zapisywania i udostępniania informacji

Imię:

Nazwisko:

Login:

Hasło:

/welcome

