Furka4U - cz. 4: edycja danych

Aplikacja Furka4U - ćw. 4

Kontynuujemy projekt serwisu ogłoszeń samochodowych. Dzisiaj dodamy:

- usuwanie ogłoszeń,
- edycję ogłoszeń.

Usuwanie ogłoszeń

- Na stronie ze szczegółami ogłoszenia powinniśmy dodać link umożliwiający usunięcie ogłoszenia.
  - W widoku `offerDetails.html`, poniżej ostatniego pola, dodaj kod z linkiem:

```
<div>
    <a th:href="|/deleteoffer/${offer.id}|" class="btn btn-danger">Usuń ogłoszenie</a>
</div>
```

- Zwróć uwagę na atrybut `th:href` - link będzie prowadził do adresu `/deleteoffer/{offer.id}`, np. `/deleteoffer/17`.
- W kontrolerze `HomeController` dodajmy metodę obsługującą usunięcie:

```
@RequestMapping("/deleteoffer/{id}")
public String deleteOffer(Model model, @PathVariable("id") Integer id) {

    return "deleteOffer";
}
```

- Musimy dodać metodę usuwającą ogłoszenie z bazy danych - w klasie `OffersService` dodaj metodę:

```
public Offer deleteOffer(Integer id) {
    Offer offer = em.find(Offer.class, id);
    em.remove(offer);
    return offer;
}
```

- Metodę tę wywołamy z poziomu kontrolera - uzupełnij kod `deleteOffer()` w klasie `HomeController`:

```
@GetMapping("/deleteoffer/{id}")
public String deleteOffer(Model model, @PathVariable("id") Integer id) {
    Offer offer = offersService.deleteOffer(id);

    model.addAttribute("offer", offer);
    return "deleteOffer";
}
```

- Pozostaje wyświetlić widok z informacją o usunięciu oferty. Stwórz nowy widok `deleteOffer.html` i umieść w nim kod:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/web/thymeleaf/layout"
      layout:decorator="layout">
    <head>
        <title>Usuwanie oferty</title>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    </head>
    <body>
        <section layout:fragment="content">
            <h2>Ogłoszenie usunięte</h2>

            <div class="row">
                <div class="col-sm-2"><b>ID oferty:</b></div>
                <div class="col-sm-8"><span th:text="${offer.id}" /></div>
            </div>
            <div class="row">
                <div class="col-sm-2"><b>Tytuł:</b></div>
                <div class="col-sm-8"><span th:text="${offer.title}" /></div>
            </div>

            <p>
                <a href="/">Powrót do listy ofert</a>
            </p>
        </section>
    </body>
</html>
```

- Sprawdź poprawność działania.

Edycja ogłoszeń

- Na stronie ze szczegółami ogłoszenia dodamy link do edycji ogłoszenia - tuż obok linka do usuwania:

```
<a th:href="|/editoffer/${offer.id}|" class="btn btn-primary">Edytuj ogłoszenie</a>
```

- Link prowadzi do adresu `/editoffer/{offer.id}`, np. `/editoffer/17`.
- W kontrolerze `HomeController` dodajmy metodę obsługującą adres `/editoffer/{id}` - jej zadaniem będzie przygotowanie formularza edycji:

```
@GetMapping("/editoffer/{id}")
public String editOffer(Model model, @PathVariable("id") Integer id) {
```

```
    return "offerForm";
}
```

- Wykorzystamy ten sam formularz, którego użyliśmy do dodawania ogłoszenia - szablon `offerForm.html`. Przyjrzyj się zawartości formularza - większość strony podczas edycji będzie identyczna, zmienić należałoby tylko:
  - tytuł strony i wyświetlany nagłówek ("Edycja ogłoszenia" zamiast "Nowe ogłoszenie"),
  - akcja, do której będzie przesłany formularz (`/editoffer` zamiast `/newoffer`),
  - formularz nie ma pola z identyfikatorem oferty - teraz będziemy go potrzebować.
- Zmodyfikujmy zatem szablon `offerForm.html` - tekst nagłówka i akcję będziemy wyświetlać dynamicznie, przekazując dane do wyświetlenia z kontrolera. W pliku szablonu:
  - atrybut `action` formularza:

    ```
    <form action="/newoffer" method="POST" th:object="${offer}" class="form-horizontal">
    ```

    zmień na:

    ```
    <form th:action="${action}" method="POST" th:object="${offer}" class="form-horizontal">
    ```

  - tytuł strony w sekcji `<head>`:

    ```
    <title>Nowe ogłoszenie</title>
    ```

    zmień na

    ```
    <title th:text="${header}"></title>
    ```

  - wyświetlany nagłówek:

    ```
    <h1>Nowe ogłoszenie</h1>
    ```

    zmień na:

    ```
    <h1 th:text="${header}"></h1>
    ```

  - dodaj ukryte pole z `id` ogłoszenia (w dowolnym miejscu wewnątrz formularza, np. tuż przed przyciskiem `submit`):

    ```
    <input type="hidden" th:field="*{id}" />
    ```

- Musimy teraz uzupełnić kod obsługujący formularz dodawania ogłoszenia w kontrolerze - tak żeby przekazał do widoku tytuł strony i akcję. W metodach `newOfferForm()` i `saveNewOffer()` dodaj kod:

  ```
  model.addAttribute("header", "Nowe ogłoszenie");
  model.addAttribute("action", "/newoffer");
  ```

- Sprawdź działanie dodawania ogłoszeń.

- Wracamy teraz do edycji ogłoszeń. W kontrolerze, w metodzie `editOffer()`, musimy podobnie ustawić tytuł strony i akcję:

```
model.addAttribute("header", "Edycja ogłoszenia");
model.addAttribute("action", "/editoffer/" + id);
```

- Podczas edycji formularz powinien zostać automatycznie wypełniony danymi ogłoszenia. Pobierzmy zatem ogłoszenie z bazy danych i przekażmy do widoku:

```
Offer offer = offersService.getOffer(id);
model.addAttribute("offer", offer);
```

- Musimy też wypełnić wszystkie listy opcji, podobnie jak przy dodawaniu nowych ogłoszeń (typy nadwozia, modele, itd.):

```
List<CarModel> carModels = offersService.getCarModels();
List<BodyStyle> bodyStyles = offersService.getBodyStyles();
List<FuelType> fuelTypes = offersService.getFuelTypes();

model.addAttribute("carModels", carModels);
model.addAttribute("bodyStyles", bodyStyles);
model.addAttribute("fuelTypes", fuelTypes);
```

- Pozostaje obsłużyć zapisanie zmienionego ogłoszenia w bazie. Zacznijmy od dodania metody w usłudze `OffersService`:
  - metoda `merge()` klasy `EntityManager` służy do zapisania zmian w encji do bazy danych; encja musi mieć wypełnione pole `id`:

```
public Offer saveOffer(Offer offer) {
    return em.merge(offer);
}
```

- Po wypełnieniu formularza edycji zostanie on przesłany pod adres `/editoffer/{id}`, metodą POST - dodajmy w kontrolerze metodę, która takie żądanie obsłuży:

```
@PostMapping("/editoffer/{id}")
public String saveEditedOffer(Model model, @PathVariable("id") Integer id, @Valid Offer

}
```

- Jeśli wystąpią błędy walidacji - chcemy pozostać na stronie z formularzem edycji:

```
@PostMapping("/editoffer/{id}")
public String saveEditedOffer(Model model, @PathVariable("id") Integer id, @Valid Offer
    if(binding.hasErrors()) {
        model.addAttribute("header", "Edycja ogłoszenia");
        model.addAttribute("action", "/editoffer/" + id);
```

```
        List<CarModel> carModels = offersService.getCarModels();
        List<BodyStyle> bodyStyles = offersService.getBodyStyles();
        List<FuelType> fuelTypes = offersService.getFuelTypes();

        model.addAttribute("carModels", carModels);
        model.addAttribute("bodyStyles", bodyStyles);
        model.addAttribute("fuelTypes", fuelTypes);

        return "offerForm";
    }


}
```

- Jeśli walidacja zakończy się pomyślnie, chcemy zapisać ofertę w bazie danych i przekierować użytkownika na stronę z danymi oferty:

```
@PostMapping("/editoffer/{id}")
public String saveEditedOffer(Model model, @PathVariable("id") Integer id, @Valid Offer
    if(binding.hasErrors()) {
        model.addAttribute("header", "Edycja ogłoszenia");
        model.addAttribute("action", "/editoffer/" + id);

        List<CarModel> carModels = offersService.getCarModels();
        List<BodyStyle> bodyStyles = offersService.getBodyStyles();
        List<FuelType> fuelTypes = offersService.getFuelTypes();

        model.addAttribute("carModels", carModels);
        model.addAttribute("bodyStyles", bodyStyles);
        model.addAttribute("fuelTypes", fuelTypes);

        return "offerForm";
    }

    offersService.saveOffer(offer);

    return "redirect:/offer/" + offer.getId();
}
```

Zadanie 1 (dodatkowe, +0.5 oceny)

- Dodaj paginację - wyświetlaj na raz nie więcej niż 20 ogłoszeń, dodaj linki do kolejnych stron.
- Podczas pobierania danych z bazy wykorzystaj metody `setFirstResult()` i `setMaxResults()` klasy `TypedQuery`.
- Pamiętaj, że aby paginacja działała poprawnie, dane muszą być zwracane

w przewidywalnej kolejności - konieczne jest sortowanie (np. po id).

Zadanie 2 (dodatkowe, +0.5 oceny)

- Dodaj możliwość sortowania ogłoszeń na liście po cenie, roku produkcji lub dacie dodania.
- W obecnym modelu danych nie ma daty dodania ogłoszenia - należałoby rozbudować tabelę `offer`, encję `Offer` oraz zmienić kod dodawania ogłoszenia, tak by automatycznie zapisywał datę. Edycja ogłoszenia nie powinna zmieniać zapisanej daty.

Zadanie 3 (dodatkowe, +1 ocena)

- Dodaj logowanie użytkowników.
- Dane użytkowników są już w bazie danych (tabela `user`). Potrzebna będzie nowa encja `User` oraz nowe pole w encji `Offer`, przechowujące użytkownika, który dodał ogłoszenie.
- W osobnym kontrolerze `UsersController` obsłuż logowanie / wylogowanie użytkownika.
- Zalogowanego użytkownika zapisz w sesji - użyj do tego komponentu o zasięgu `@SessionScope`.
- Przy dodawaniu ogłoszenia zapisuj automatycznie, kto dodał ogłoszenie.
- Nie pozwól użytkownikowi usuwać ani edytować cudzych ogłoszeń.

Zadanie 4 (dodatkowe, +0.5 oceny)

- Dodaj możliwość wyszukiwania ogłoszeń po fragmencie opisu.

Zadanie 5 (dodatkowe, +0.5 oceny)

- Zmień formularz edycji i dodawania ogłoszenia:
  - rozdziel model i markę pojazdu;
  - po wybraniu marki dynamicznie powinna załadować się lista modeli.
- Zadanie można zrealizować za pomocą JavaScript i np. jQuery (metody `$.ajax()`, `$.get()` lub `load()`).
- Po stronie serwera - potrzebna będzie metoda w kontrolerze, zwracająca samą listę modeli (w postaci JSON lub fragmentu HTML). Wykorzystaj adnotację `@ResponseBody`, żeby ominąć system szablonów Thymeleaf i bezpośrednio zwrócić treść odpowiedzi z kontrolera.