Furka4U - cz. 3: filtrowanie i edycja danych

Aplikacja Furka4U - ćw. 3

Kontynuujemy projekt serwisu ogłoszeń samochodowych. Dzisiaj dodamy:

- wyszukiwanie / filtrowanie ogłoszeń,
- dodawanie nowych ogłoszeń.

Filtrowanie ofert

- Na stronie głównej dodamy możliwość filtrowania ofert po marce i modelu.
- W widoku `offersList.html`, na samym początku sekcji, dodaj kod z formularzem filtrowania:

```
<div id="offerFilter" class="panel panel-default">
    <div class="panel-heading">Szukaj ofert</div>
    <div class="panel-body">
        <form id="offerFilterForm" action="/" method="get" class="form-inli
            <select class="form-control">
                <option value="">Marka</option>
            </select>
            <select class="form-control">
                <option value="">Model</option>
            </select>

            <input type="submit" value="Filtruj" class="form-control"/>
        </form>
    </div>
</div>
```

- W formularzu mamy dwie listy rozwijane - w pierwszej musimy wyświetlić listę producentów, w drugiej - listę modeli dla wybranego producenta. Zajmijmy się najpierw pierwszą z list.
- W kontrolerze już wcześniej przygotowaliśmy listę producentów i przekazaliśmy do widoku jako atrybut `carManufacturers`. Dodajmy zatem opcje do pierwszej listy rozwijanej - wewnątrz pierwszego z elementów `select`, po opcji reprezentującej brak wyboru marki (`<option value="">Marka</option>`), dodaj kod iterujący po liście producentów i wyświetlający opcję dla każdego z nich:

```
<option th:each="manufacturer : ${carManufacturers}" th:value="${manufacturer.id}" th:t
```

- Aby obsłużyć formularz, przygotujmy klasę `OfferFilter`. Na razie zawierać będzie tylko pola z producentem i modelem, ale docelowo może reprezentować bardziej złożony filtr, np. z zakresem cen czy lat produkcji.
  - Stwórz klasę `OfferFilter` w pakiecie `wizut.tpsi.ogloszenia.web`.

  - Dodaj do niej pola `manufacturerId` i `modelId`, oba typu `Integer`. Wygeneruj gettery i settery.

- Obiekt klasy `OfferFilter` dodamy do listy argumentów metody `home()` w kontrolerze `HomeController`:

```
@GetMapping("/")
public String home(Model model, OfferFilter offerFilter) {
    ...
}
```

- Formularz na stronie `offersList.html` powiążemy z obiektem klasy `OfferFilter`, dodając atrybuty `th:object` i `th:field`:

  - do elementu `<form>` dodaj atrybut:

```
th:object="${offerFilter}"
```

  - do elementu `<select>` dodaj atrybut:

```
th:field="*{manufacturerId}"
```

- Musimy teraz obsłużyć filtrowanie ofert po stronie kontrolera. Przed pobraniem listy ofert sprawdzimy, czy mamy ustawiony parametr `manufacturerId` - jeśli tak, pobierzemy oferty dla danego producenta.

  - Zastąp linijkę:

```
List<Offer> offers = offersService.getOffers();
```

  następującym kodem:

```
List<Offer> offers;

if(offerFilter.getManufacturerId()!=null) {
    offers = offersService.getOffersByManufacturer(offerFilter.getManufacturerId());
} else {
    offers = offersService.getOffers();
}
```

Dodawanie ogłoszeń

- Zaimplementujemy teraz funkcję dodawania nowych ogłoszeń. W tym celu:
  - dodamy metodę w kontrolerze, zmapowaną na url `/newoffer`, która spowoduje wyświetlenie formularza,
  - przygotujemy formularz dodawania ogłoszenia `offerForm.html`,
  - dodamy metodę w kontrolerze, obsługującą dane przesłane z formularza,
  - dodamy metodę w usłudze `OffersService`, służącą do zapisania nowej oferty w bazie danych.

Formularz dodawania ogłoszenia

- W klasie `HomeController` dodaj metodę `newOfferForm()`:

```
@GetMapping("/newoffer")
public String newOfferForm(Model model, Offer offer) {
    return "offerForm";
}
```

- Pora na widok - dodaj do projektu nowy szablon o nazwie `offerForm.html`.
  Umieść w nim następujący 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>Nowe ogłoszenie</title>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    </head>
    <body>
        <section layout:fragment="content">
            <h1>Nowe ogłoszenie</h1>

        </section>
    </body>
</html>
```

- Wewnątrz `<section>` umieścimy formularz z danymi ogłoszenia:
  - wykorzystamy klasy CSS z biblioteki Bootstrap (patrz http://getbootstrap.com/css/#forms);
  - pola formularza powinny odpowiadać własnościom klasy `Offer`;
  - formularz powinien zostać wysłany pod adres `/newoffer` metodą `POST`.

  Kod formularza:

```
<form action="/newoffer" method="POST" class="form-horizontal">
    <div class="form-group">
        <label class="col-sm-2 control-label">Tytuł:</label>
        <div class="col-sm-10">
            <input type="text" class="form-control"/>
        </div>
    </div>
    <div class="form-group">
        <label class="col-sm-2 control-label">Rocznik:</label>
        <div class="col-sm-4">
            <input type="number" class="form-control"/>
```

```
    </div>
    <label class="col-sm-2 control-label">Przebieg:</label>
    <div class="col-sm-4">
        <input type="number" class="form-control"/>
    </div>
</div>
<div class="form-group">
    <label class="col-sm-2 control-label">Pojemność silnika:</label>
    <div class="col-sm-4">
        <input type="number" step="0.1" class="form-control"/>
    </div>
    <label class="col-sm-2 control-label">Moc silnika:</label>
    <div class="col-sm-4">
        <input type="number" class="form-control"/>
    </div>
</div>
<div class="form-group">
    <label class="col-sm-2 control-label">Liczba drzwi:</label>
    <div class="col-sm-4">
        <input type="number" class="form-control"/>
    </div>
    <label class="col-sm-2 control-label">Kolor:</label>
    <div class="col-sm-4">
        <input type="text" class="form-control"/>
    </div>
</div>
<div class="form-group">
    <label class="col-sm-2 control-label">Model:</label>
    <div class="col-sm-10">
        <select class="form-control">
        </select>
    </div>
</div>
<div class="form-group">
    <label class="col-sm-2 control-label">Nadwozie:</label>
    <div class="col-sm-4">
        <select class="form-control">
        </select>
    </div>
    <label class="col-sm-2 control-label">Rodzaj paliwa:</label>
    <div class="col-sm-4">
        <select class="form-control">
        </select>
    </div>
</div>
<div class="form-group">
```

```
            <label class="col-sm-2 control-label">Cena:</label>
            <div class="col-sm-4">
                <input type="number" class="form-control"/>
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-2 control-label">Opis:</label>
            <div class="col-sm-10">
                <textarea class="form-control" rows="5"/>
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-4"></div>
            <div class="col-sm-4">
                <input type="submit" value="Zapisz" class="form-control btn btn-primary"/>
            </div>
            <div class="col-sm-4">
                <a href="/" class="btn btn-default">Anuluj</a>
            </div>
        </div>
    </form>
```

- Musimy teraz powiązać pola formularza z własnościami obiektu klasy `Offer`:
  - do elementu `<form>` dodaj atrybut `th:object="${offer}"`,
  - do wszystkich pól `<input>` i `<textarea>` dodaj odpowiednie atrybuty `th:field`, np. dla pola "Tytuł:":

```
<input type="text" th:field="*{title}" class="form-control"/>
```

  - do wszystkich pól `<select>` dodaj odpowiednie atrybuty `th:field` - np. dla pola "Nadwozie:":

```
<select th:field="*{bodyStyle.id}" class="form-control">
```

  Ponieważ własność `bodyStyle` jest obiektem (encją zawierającą własności `id` i `name`), pole formularza mapujemy nie na własność `bodyStyle`, a na jego id `bodyStyle.id`. Analogiczny kod dodaj dla pól `model` i `fuelType`.

- Jak widać, listy rozwijane (lista modeli, typy nadwozia, typ paliwa) nie są jeszcze wypełnione. Musimy przekazać je z kontrolera i wyświetlić w widoku. W kontrolerze `HomeController`, w metodzie `newOfferForm` dodaj kod pobierający wszystkie listy z bazy i przekazujący je do widoku:

```
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);
```

- W widoku wypełnijmy listę typów nadwozia opcjami - wewnątrz `<select>` dla pola `bodyStyle` dodaj kod:

```
<option th:each="bs : ${bodyStyles}" th:value="${bs.id}" th:text="${bs.name}"></option>
```

- Sprawdź działanie formularza. Lista wariantów nadwozia powinna już być wypełniona.

Zadanie 1

- Wypełnij opcjami listę `fuelType`. Sprawdź działanie.
- Przy wyborze modelu samochodu docelowo chcemy mieć możliwość wyboru marki, a następnie z przefiltrowanej listy modeli - modelu. Na razie jednak przyjmiemy uproszczone podejście. Wyświetlimy listę wszystkich modeli, bez podziału na markę, np:

```
Ford Fiesta
Ford Focus
Toyota Avensis
Toyota Corolla
```

- Wypełnij opcjami listę modeli. Wyświetlając opis modelu chcemy pokazać nazwę marki i nazwę modelu.

Zapis ogłoszenia do bazy

- Dodamy teraz w klasie `OffersService` metodę zapisującą nową ofertę w bazie danych:
  - do zapisania nowej encji służy metoda `EntityManager.persist()`;
  - po zapisaniu zwracamy obiekt encji jako wynik metody - mógł on zostać zmieniony podczas zapisywania do bazy, np. uzupełniony o automatycznie nadane id.

```
public Offer createOffer(Offer offer) {
    em.persist(offer);
    return offer;
}
```

- Ponieważ nasza usługa `OffersService` obejmuje teraz zapis danych do bazy, musimy włączyć w niej obsługę transakcji. Do klasy `OffersService` dodajmy adnotację `@Transactional`:

```
@Service
@Transactional
public class OffersService {
```

- Musimy teraz dodać w kontrolerze metodę obsługującą wysłany formularz. Zmapowana będzie na ten sam url `/newoffer`, jednak obsługiwać będzie metodę `POST`:
  - zwróć uwage na parametry metody - `@Valid` spowoduje walidację obiektu `offer`, `BindingResult` przechowa informacje o wynikach walidacji;
  - jeśli wystąpią błędy walidacji - ponownie pokazujemy formularz `offerForm`;
  - jeśli nie znaleziono błędów - zapisujemy ogłoszenie do bazy i przekierowujemy użytkownika na stronę z nowododanym ogłoszeniem.

```
@PostMapping("/newoffer")
public String saveNewOffer(Model model, @Valid Offer offer, BindingResult binding) {
    if(binding.hasErrors()) {
        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";
    }
    offer = offersService.createOffer(offer);

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

- Aby walidacja działała poprawnie, musimy dodać odpowiednie adnotacje do klasy `Offer`. Ograniczymy zakres pewnych pól, wszystkie wymagane pola oznaczymy adnotacją `@NotNull`, itp.
  - Dodaj adnotację `@NotNull` do pól: `title`, `year`, `mileage`, `doors`, `colour`, `description`, `price`, `model`, `bodyStyle`, `fuelType`.
  - Pole `title` - narzucimy ograniczenia na długość, od 5 do 255 znaków: `@Size(max = 255, min = 5)`
  - Podobne ograniczenia dodajmy do pól `colour` (od 3 do 30) oraz `description` (od 5 do 65535)
  - Pole `rok` - nie spodziewamy się pojazdu starszego niż rok 1900: `@Min(1900)`
  - Pola `mileage`, `engineSize`, `enginePower`, `price` - wartości ujemne nie mają tu sensu: `@Min(0)`

  - Pole `doors` - zakładając, że nie sprzedajemy autobusów, możemy ograniczyć wartości do przedziału <1, 5>:

```
@Min(1)
@Max(5)
```

- Sprawdź działanie formularza - dopóki nie wpiszemy poprawnie wszsytkich danych, nie wyjdziemy z formularza. Po poprawnym wypełnieniu, ogłoszenie powinno zostać zapisane.
- Brakuje nam jeszcze wyświetlania informacji o błędach walidacji. Dodajmy klasy CSS, które spowodują podświetlenie błędnie wypełnionych pól.
  - framework Bootstrap zdefiniował klasę `has-error`; dodanie takiej klasy do pola formularza spowoduje oznaczenie go czerwoną ramką;
  - klasę `has-error` należy dodać nie do samego pola, a do elementu nadrzędnego (np. `<div>` otaczający pole `<input>`);
  - wykorzystamy atrybut Thymeleaf `th:classappend` aby dodać klasę `has-error` do błędnych elementów - musimy umieścić go na każdym z elementów `div` zawierających jakieś pole;
  - aby sprawdzić, czy dane pole jest błędne, użyjemy funkcji Thymeleaf `#fields.hasError()`;
  - przykład - dla pola tytuł docelowy kod będzie wyglądał tak:

```
<div class="form-group">
    <label class="col-sm-2 control-label">Tytuł:</label>
    <div class="col-sm-10" th:classappend="${#fields.hasErrors('*{title}')?'has-error':
        <input type="text" th:field="*{title}" class="form-control"/>
    </div>
</div>
```

Zadanie 2

- Dodaj wyświetlanie błędów walidacji do wszystkich pozostałych pól.

Zadanie 3 (dodatkowe, +0.5 oceny)

- Dodaj filtrowanie po modelu: listę modeli wyświetlaj tylko wtedy, gdy wybrano już producenta; pobierz wówczas modele wybranego producenta i tylko te modele wyświetl na liście wyboru:
  - w kontrolerze sprawdzaj, czy wybrano już model i producenta:
    - jeśli wybrano model:
      - pobierz ogłoszenia pasujące do wybranego modelu;
      - pobierz modele pasujące do wybranego producenta;
    - jeśli nie wybrano modelu, ale wybrano producenta:
      - pobierz ogłoszenia pasujące do wybranego producenta;
      - pobierz modele pasujące do wybranego producenta;

        * w pozostałych przypadkach:
            · pokaż pełną listę ogłoszeń;
            · nie pobieraj listy modeli.
- Przykład działania:
  - krok 1 - brak filtra:
    * na liście producentów widzimy wszystkich producentów;
    * lista modeli jest pusta;
    * widzimy wszystkie ogłoszenia;
  - krok 2 - wybrano producenta 'Ford', kliknięto 'Filtruj':
    * na liście producentów widzimy wszystkich producentów, wybrany 'Ford';
    * lista modeli wypełniona modelami Forda;
    * widzimy ogłoszenia sprzedaży Fordów;
  - krok 3 - wybrano model 'Focus', kliknięto 'Filtruj':
    * na liście producentów widzimy wszystkich producentów, wybrany 'Ford';
    * lista modeli wypełniona modelami Forda;
    * widzimy ogłoszenia sprzedaży Forda Focus.

Zadanie 4 (dodatkowe, +0.5 oceny)

- Dodaj filtrowanie po roku produkcji (od - do) i rodzaju paliwa. Dodaj niezbędne pola w formularzu oraz klasie `OfferFilter`.
- Do tej pory do pobierania ofert używaliśmy trzech różnych metod, w zależności od tego, czy chcieliśmy pobrać wszystkie oferty, czy też filtrować po producencie lub po modelu. Teraz pora uogólnić mechanizm wyszukiwania, tak żeby był w stanie obsłużyć dowolny zestaw parametrów wyszukiwania (np. szukanie po modelu i roku, albo po producencie i cenie, itp).
- Konieczne będzie dynamiczne budowanie zapytania JPQL:
  - w serwisie dodaj metodę `getOffers(OfferFilter filter)`;
  - sprawdzaj dla każdego pola z klasy `OfferFilter`, czy ma wartość różną od `null`; jeśli tak, dodaj do zapytania odpowiedni warunek.