Furka4U - cz. 2: wykonywanie zapytań

Aplikacja Furka4U - ćw. 2

Kontynuujemy projekt serwisu ogłoszeń samochodowych. Dzisiaj dodamy:

- layout z biblioteką Bootstrap,
- pozostałe encje JPA,
- wyświetlanie listy ogłoszeń,
- wyświetlanie szczegółów ogłoszenia.

Layout strony

- Stwórz nowy plik `layout.html` w folderze z szablonami Thymeleaf (`src/main/resources/templates`)
- W pliku zdefiniujemy layout strony, nagłówek z nawigacją oraz załadujemy CSS z biblioteki Bootstrap:

```
<!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">
    <head>
        <title layout:title-pattern="Furka4U - $CONTENT_TITLE">Furka4U</title>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />

        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <img src="https://upload.wikimedia.org/wikipedia/commons/2/23/BlondeWom

                <h1>Furka4U <small>Najlepsze furki w necie</small></h1>
                <h4>
                <a href="/">Przeglądaj ogłoszenia</a> | <a href="/newoffer">Dodaj ogłos
                </h4>
            </div>

            <div layout:fragment="content">
                Tu będzie wstawiona właściwa zawartość widoku.
            </div>
        </div>
    </body>
</html>
```

- Na głównej stronie będziemy wyświetlać listę ogłoszeń, posłuży nam do tego widok `offersList.html` - dodaj go do projektu:

```
<!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>Lista ofert</title>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    </head>
    <body>
        <section layout:fragment="content">
        </section>
    </body>
</html>
```

- Stworzony na poprzednich zajęciach widok `home.html` możesz usunąć.
- W kontrolerze `HomeController` kod metody `home()` możesz usunąć, jako wynik metody zwróć nazwę widoku `offersList`:

```
@GetMapping("/")
public String home(Model model) {
    return "offersList";
}
```

- W dalszej części ćwiczenia uzupełnimy tę metodę - pobierzemy z bazy danych listę ofert i wyświetlimy w widoku `offersList`.

Encje JPA

- Na poprzednich zajęciach stworzyliśmy dwie klasy encji: `CarManufacturer` i `CarModel`. Pora uzupełnić projekt o pozostałe encje: `BodyStyle`, `FuelType`, `Offer`.
- Wzorując się na encji `CarManufacturer` samodzielnie przygotuj encje `BodyStyle` i `FuelType`. Obie zawierać będą tylko pola `id` i `name`, podobnie jak `CarManufacturer`. Umieść je w tym samym pakiecie, co `CarManufacturer` i `CarModel` (czyli `wizut.tpsi.ogloszenia.jpa`)
- Przygotujmy encję `Offer` - dodaj do pakietu `wizut.tpsi.ogloszenia.jpa` nową klasę `Offer`.
- W klasie `Offer` umieść własności odpowiadające kolumnom w bazie danych:

```
    private Integer id;

    private String title;
```

```
    private Integer year;

    private Integer mileage;

    private BigDecimal engineSize;

    private Integer enginePower;

    private Integer doors;

    private String colour;

    private String description;

    private Integer price;

    private CarModel model;

    private BodyStyle bodyStyle;

    private FuelType fuelType;
```

Zwróć uwagę na pola `model`, `bodyStyle` i `fuelType` - zamiast samych identyfikatorów typu `int`, mamy tu pełne obiekty, czyli relacje do encji `CarModel`, `BodyStyle` i `FuelType`.

- Wygeneruj gettery, settery i bezargumentowy konstruktor (prawy klawisz -> **Insert code**)
- Dodaj wymagane adnotacje JPA:
  - dla klasy - `@Entity` i `@Table`:

```
@Entity
@Table(name = "offer")
public class Offer {
```

  - dla pól:
    - dla wszystkich - `@Column`;
    - dla pól tekstowych - `Size`;
    - dla pola `description`, które w bazie jest typu `TEXT/CLOB` - `Lob`;
    - dla relacji do innych encji - `ManyToOne` i `JoinColumn`:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id")
private Integer id;
```

```
@Size(max = 255)
@Column(name = "title")
private String title;

@Column(name = "year")
private Integer year;

@Column(name = "mileage")
private Integer mileage;

@Column(name = "engine_size")
private BigDecimal engineSize;

@Column(name = "engine_power")
private Integer enginePower;

@Column(name = "doors")
private Integer doors;

@Size(max = 30)
@Column(name = "colour")
private String colour;

@Lob
@Size(max = 65535)
@Column(name = "description")
private String description;

@Column(name = "price")
private Integer price;

@JoinColumn(name = "model_id", referencedColumnName = "id")
@ManyToOne
private CarModel model;

@JoinColumn(name = "body_style_id", referencedColumnName = "id")
@ManyToOne
private BodyStyle bodyStyle;

@JoinColumn(name = "fuel_type_id", referencedColumnName = "id")
@ManyToOne
private FuelType fuelType;
```

Pobieranie danych z bazy

- W klasie `OffersService` mamy przygotowane dwie metody: `getManufacturer()` i `getModel()`. Obie pobierają z bazy danych pojedynczą encję na podstawie wartości klucza głównego `id`. Przygotujemy teraz metody zwracające całe listy obiektów.
- Dodaj metodę `getCarManufacturers()` zwracającą listę obiektów `CarManufacturer`:

  ```
  public List<CarManufacturer> getCarManufacturers() {

  }
  ```

- Wykorzystamy zapytania w języku JPQL, podobnym do SQL, jednak operujacym na encjach JPA zamiast na tabelach bazy danych. Najprostsze zapytanie o listę wszystkich producentów będzie wyglądało tak:

  ```
  select cm from CarManufacturer cm order by cm.name
  ```

- Do wykonania zapytania użyjemy wstrzykniętego obiektu klasy `EntityManager`:

  ```
  public List<CarManufacturer> getCarManufacturers() {
      String jpql = "select cm from CarManufacturer cm order by cm.name";
      TypedQuery<CarManufacturer> query = em.createQuery(jpql, CarManufacturer.class);
      List<CarManufacturer> result = query.getResultList();
      return result;
  }
  ```

- W powyższym kodzie:
  - do metody `createQuery()` przekazujemy string z zapytaniem oraz klasę, której spodziewamy się w wyniku zapytania;
  - metoda `getResultList()` zwraca od razu listę obiektów wskazanej uprzednio klasy - tutaj: `CarManufacturer`.
- W przypadku prostych zapytań ten sam kod można zapisać znacznie zwięźlej:

  ```
  return em.createQuery("select cm from CarManufacturer cm order by cm.name", CarManufact
  ```

  Zostawmy jednak pierwotną postać, aby mieć wzór dla bardziej złożonych zapytań, które będziemy konstruować za chwilę.
- Przetestujmy działanie nowej metody - w kontrolerze `HomeController` pobierzemy listę producentów i przekażemy do wyświetlenia w widoku:

  ```
  public String home(Model model) {
      List<CarManufacturer> carManufacturers = offersService.getCarManufacturers();

      model.addAttribute("carManufacturers", carManufacturers);
  ```

```
    return "offersList";
}
```

- W widoku `offersList.html` tymczasowo wyświetlimy listę producentów - wewnątrz elementu `<section>` dodaj kod:

```
<p th:each="cm : ${carManufacturers}" th:text="${cm.name}"></p>
```

Zadanie 1

- Przygotuj metody pobierające listę rodzajów nadwozia (metoda `getBodyStyles()`), rodzajów paliwa (metoda `getFuelTypes()`), modeli (metoda `getCarModels()`).
- Przetestuj dodane metody - wyświetl na stronie wszystkie listy.

Pobieranie danych - zapytania z parametrami

- Dodamy teraz metodę pobierającą listę modeli samochodów danego producenta. Nowa metoda `getCarModels()` będzie przyjmowała jako argument identyfikator producenta:

```
public List<CarModel> getCarModels(int manufacturerId) {
    String jpql = "select cm from CarModel cm where cm.manufacturer.id = :id order by c

    TypedQuery<CarModel> query = em.createQuery(jpql, CarModel.class);
    query.setParameter("id", manufacturerId);

    return query.getResultList();
}
```

- W zapytaniu JPQL pojawia się *placeholder* na parametr o nazwie `id`. Wartość parametru ustawiamy dwie linijki niżej, za pomocą metody `query.setParameter()`.

Zadanie 2

- Dodaj metodę zwracającą listę wszystkich ogłoszeń: `getOffers()`.
- Wzorując się na metodzie `getCarModels(int manufacturerId)` przygotuj dwa nowe warianty metody:
  - `getOffersByModel(int modelId)` - zwracającą listę ofert sprzedaży określonego modelu samochodu,
  - `getOffersByManufacturer(int manufacturerId)` - zwracającą listę ofert sprzedaży określonej marki.
- Wzorując się na metodzie `getManufacturer(int id)` przygotuj metodę `getOffer(int id)` zwracającą pojedynczą ofertę na podstawie jej id.

Wyświetlanie listy ofert na stronie głównej

- W kontrolerze pobierzemy listę wszystkich ofert i przekażemy ją do wyświetlenia. Pobierzemy też listę producentów i modeli - przydadzą się później do formularza wyszukiwania ofert.
- Kod metody `home()` w kontrolerze `HomeController` zastąp następującym:

```
public String home(Model model) {
    List<CarManufacturer> carManufacturers = offersService.getCarManufacturers();
    List<CarModel> carModels = offersService.getCarModels();

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

    model.addAttribute("carManufacturers", carManufacturers);
    model.addAttribute("carModels", carModels);
    model.addAttribute("offers", offers);

    return "offersList";
}
```

- W widoku `offersList` wyświetlimy listę ofert w tabelce - wyczyść zawartość `<section>` i zastąp kodem:

```
<table class="table">
    <tr>
        <th>Tytuł</th>
        <th>Marka</th>
        <th>Model</th>
        <th>Rocznik</th>
        <th>Cena</th>
    </tr>
    <tr th:each="offer : ${offers}">
        <td th:text="${offer.title}"></td>
        <td th:text="${offer.model.manufacturer.name}"></td>
        <td th:text="${offer.model.name}"></td>
        <td th:text="${offer.year}"></td>
        <td th:text="${offer.price}"></td>
    </tr>
</table>
```

- Sformatujmy porządnie kolumnę z ceną:
  - dodajmy klasę Bootstrap `text-right`,
  - dodajmy walutę,
  - dodajmy formatowanie liczby z separatorem tysięcy:

```
<td class="text-right">
    <span th:text="${#numbers.formatDecimal(offer.price, 0, 'WHITESPACE', 0,'COMMA'
```

```
        zł
    </td>
```

Wyświetlanie szczegółów oferty

- Do tabelki dodamy kolumnę z linkiem **Pokaż**, prowadzącym do strony z pełnym opisem oferty.
- W widoku `offersList` dodajmy nową kolumnę - w wierszu nagłówkowym pustą komórkę:

  ```
  <th></th>
  ```

  a w wierszach z danymi - komórkę z linkiem:

  ```
  <td><a th:href="|/offer/${offer.id}|">Pokaż</a></td>
  ```

- Link prowadzi do url z id oferty, np. `/offer/7`. Dodajmy w kontrolerze `HomeController` metodę obsługującą taki url, a w niej:
  - wyciągamy id z adresu,
  - pobieramy ofertę z bazy danych używając `offersService.getOffer()`,
  - przekazujemy ofertę do widoku do wyświetlenia,
  - zwracamy nazwę widoku `offerDetails`:

  ```
  @GetMapping("/offer/{id}")
  public String offerDetails(Model model, @PathVariable("id") Integer id) {
      Offer offer = offersService.getOffer(id);
      model.addAttribute("offer", offer);
      return "offerDetails";
  }
  ```

- Pozostaje przygotować widok wyświetlający szczegóły oferty. Stwórz plik `offerDetails.html`, 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 th:text="${offer.title}">Szczegóły oferty</title>
          <meta charset="UTF-8" />
          <meta name="viewport" content="width=device-width, initial-scale=1.0" />
      </head>
      <body>
          <section layout:fragment="content">
              <h1 th:text="${offer.title}"></h1>
              <h2>
                  Cena:
                  <span class="text-primary" th:text="${offer.price}"></span>
  ```

```
            </h2>

            <table th:object="${offer}" class="table">
                <tr>
                    <th>Marka:</th>
                    <td th:text="*{model.manufacturer.name}"></td>
                    <th>Nadwozie:</th>
                    <td th:text="|*{bodyStyle.name} *{doors}D|"></td>
                </tr>
                <tr>
                    <th>Model:</th>
                    <td th:text="*{model.name}"></td>
                    <th>Kolor:</th>
                    <td th:text="*{colour}"></td>
                </tr>
            </table>
            <div th:text="${offer.description}" class="well">
            </div>
        </section>
    </body>
</html>
```

Zadanie 3

- Dodaj wyświetlanie pozostałych pól opisujących ofertę - rok produkcji, przebieg, pojemność i moc silnika, rodzaj paliwa.
- Sformatuj wyświetlanie ceny: dodaj walutę i separator tysięcy.