Programowanie komponentowe Spring

Ćw5 - JDBC

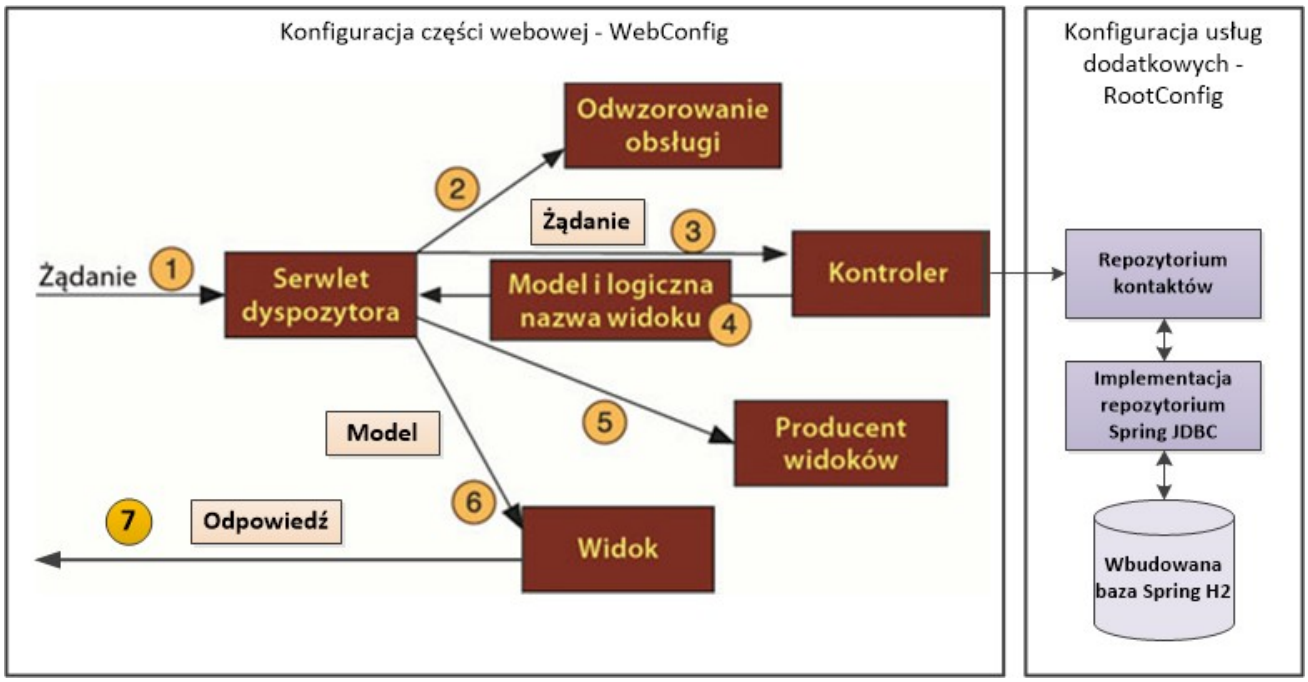
Wstęp

Laboratorium nr 5 składa się z zadań umożliwiających:

- utworzenie repozytorium jdbc, korzystające z tymczasowej bazy danych umieszczonej w pamięci
- utworzenie kontrolera i formularza do dopisywania informacji do bazy danych
- utworzenia podstawowej walidacji pól formularza
- edycji i usunięcia kontaktu.

Zadanie 4

Celem zadania 4 jest skonfigurowanie podstawowego źródła danych - repozytorium JDBC. Architektura aplikacji MVC z repozytorium JDBC została pokazana na rysunku.



5.1 Repozytorium kontaktów JDBC

Konfiguracja źródła danych i szablonu JDBC

W celu konfiguracji bazy danych i dostępu JDBC należy utworzyć klasę konfiguracyjną DataConfig w pakiecie **contacts.config**, która zostanie automatyczne odszukana przez klasę konfiguracyjną RootConfig. Klasa DataConfig jest oznaczona adnotacją @**Configuration** i zawiera definicję dwóch komponentów:

- komponent - źródło danych

```
@Bean
public DataSource dataSource() {
  return new EmbeddedDatabaseBuilder()
        .setType(EmbeddedDatabaseType.H2)
        .addScript("schema.sql") // import schematu bazy danych
```

```
        .addScript("data.sql")  // import danych testowych
        .build();
    }
```

- komponent - szablon JDBC z wstrzykniętym źródłem danych

```
@Bean
public JdbcOperations jdbcTemplate(DataSource dataSource) {
  return new JdbcTemplate(dataSource);
}
```

Przechowywanie kontaktów w tabeli

W celu przechowywania danych kontaktów w tabeli bazy danych, kontakt powinien dodatkowo posiadać unikalny identyfikator, który będzie indeksem głównym tabeli.

- właściwości klasy Contact

```
public class Contact {
private Long id;
private String firstName;
private String lastName;
private String email;
private String phone;
```

- konstruktor uwzględniający parametr id kontaktu
- getter i setter dla id kontaktu

Implementacja interfejsu JdbcContactRepository

Implementacja interfejsu zawiera metody wykorzystujące wstrzyknięty szablon jdbc do wykonywania instrukcji SQL na bazie danych:

- **findContacts** - wyszukanie listy kontaktów do wyświetenia na stronie home.html

```
@Override
public List<Contact> findContacts(long max, int count) {
return jdbc.query("SELECT id, first_name, last_name, email, phone" +
" FROM Contact" +
" WHERE id > ?" +
" ORDER by id asc limit 20",
  new ContactRowMapper(), max);
}
```

- **mapper** - prywatna statyczna klasa wewnętrzna implementująca metodę mapującą odczytany wiersz tabeli z bazy danych na obiekt klasy **Contact**

```
private static class ContactRowMapper implements RowMapper<Contact> {
public Contact mapRow(ResultSet rs, int rowNum) throws SQLException {
return new Contact(rs.getLong("id"),
rs.getString("first_name"),
```

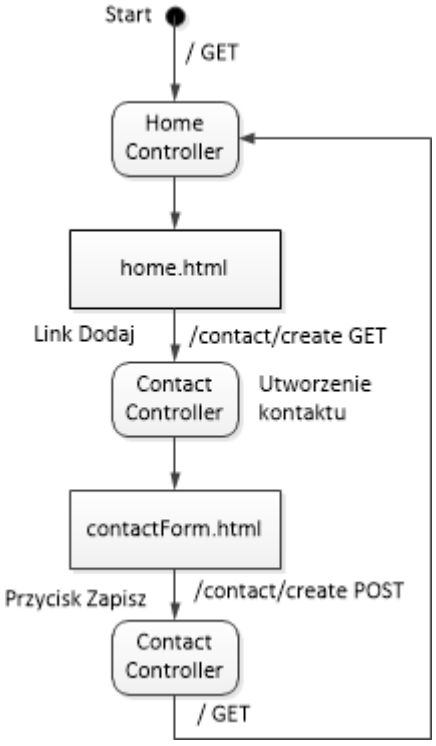
```
rs.getString("last_name"),
rs.getString("email"),
rs.getString("phone"));
}
}
```

Treść zadania 4

Proszę zaimportować projekt contacts4. Następnie skonfigurować źródło danych, rozszerzyć klasę Contact i utworzyć implementację interfejsu dla JDBC. Dodatkową właściwość ID proszę wyświetlić w tabeli w widoku home.html.

Zadanie 5

Celem zadania 5 jest dodanie w aplikacji contacts możliwości dopisywania kontaktów do bazy danych. Dodatkowa funkcjonalność wymaga utworzenia kontrolera kontaktów i formularza do wprowadzania danych. Docelowy przepływ żądań między widokami i kontrolerami jest pokazany na Rys. 5.2.



Rys 5.2 Przepływ żądań między widokami i kontrolerami przy dodawaniu kontaktu

Rozszerzenie interfejsu ContactRepository

Dla potrzeb kolejnych zadań interfejs repozytorium został rozszerzony o metodę:

- **insert** - zapisanie nowo dodanego kontaktu

Implementacja interfejsu JdbcContactRepository

Implementacja interfejsu zawiera metody wykorzystujące wstrzyknięty szablon jdbc do wykonywania instrukcji SQL na bazie danych:

- **insert** - zapisanie nowo dodanego kontaktu

```
@Override
public void insert(Contact contact) {
jdbc.update("INSERT INTO Contact "
+ "(first_name, last_name, email, phone)"
+ " VALUES (?, ?, ?, ?)",
contact.getFirstName(),
contact.getLastName(),
contact.getEmail(),
contact.getPhone());
```

Kontroler kontaktów ContactController

Kontroler kontaktów obsługuje żądania o ścieżkach rozpoczynające się słowa “/contact”

- ustawienie obsługiwanej ścieżki

```
@Controller
@RequestMapping("/contact")
public class ContactController {
```

  - wstrzyknięcie odwołania do repozytorium kontaktów

```
@Autowired
private ContactRepository contactRepository;
```

- metoda GET jest wywoływana po kliknięciu na linku Dodaj w widoku home.html - tworzy nowy kontakt i dodaje go do modelu

```
    @RequestMapping(value = "/create", method = GET)
public String showRegistrationForm(Model model) {
model.addAttribute(new Contact());
return "contactForm";
}
```

- metoda POST jest wywoływana po kliknięciu przycisku Zapisz w widoku contactForm.html - zapisuje dane z formularza do repoztorium

```
    @RequestMapping(value = "/create", method = POST)
public String processContact(Contact contact) {
contactRepository.insert(contact);
return "redirect:/";
}
```

Link w widoku home.html

Widok powinien umożliwiać dodanie nowego kontaktu.

- link do tworzenia nowego kontaktu

```
<a th:href="@{/contact/create}">Dodaj kontakt</a>
```

Widok formularza kontaktów contactForm.html

- **element form** - wszystkie pola wejściowe input i przycisk Zapisz powinny być umieszczone wewnątrz elementu <form>, który po wciśnięciu przycisku powoduje wysłanie żądania POST
  <form method="POST" th:object="${contact}">
- **tabela** - etykiety pól formularza i pola powinny być umieszczone w wierszach i kolumnach tabeli. Wartość pola id jest nadawana automatycznie, więc nie powinna być wprowadzana.
  <table>
  <tr>
  <td>Imię</td>
  <td><input type="text" th:field="*{firstName}"/></td>
  </tr>
  ...
- **przycisk Zapisz** - wciśnięcie przycisku powoduje wysłanie żądania **/contact/create POST** do dyspozytora
  <td><input type="submit" value="Zapisz" /></td>

Treść zadania 5

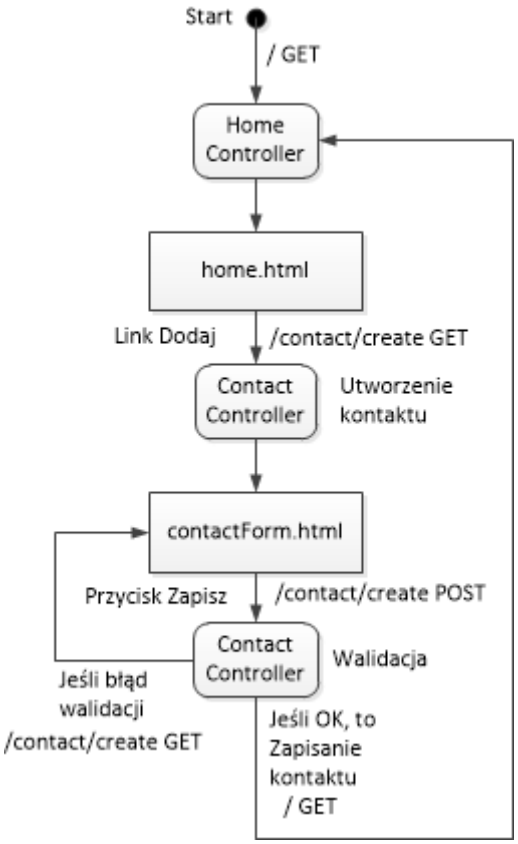
Proszę zaimportować projekt **contacts5** lub

- skopiować folder projektu **contacts4**, do folderu **contacts5**,
- zmienić nazwę projektu w pliku **.project** na **contacts5**,
- usunąć folder **.settings** i folder **bin** w folderze **contacts5**,
- zaimportować projekt eclipse z folderu **contacts5**.

Następnie rozszerzyć interfejs i implementację repozytorium kontaktów, utworzyć kontroler kontaktów, dodać link w widoku **home.html** i utworzyć formularz **contactForm.html**.

Zadanie 6

Celem zadania 6 jest wykonanie walidacji danych wprowadzonych w formularzu. Walidacja powinna przeprowadzona w obsłudze żądania **/contact/create POST** w kontrolerze **ContactController** na podstawie adnotacji umieszczonych we właściwościach klasy **Contact**. Docelowy przepływ żądań między widokami i kontrolerami jest pokazany na Rys. 5.3.



Rys 5.3 Przepływ żądań i widoków przy dodawaniu kontaktu z walidacją danych

Dodanie reguł walidacji pól - adnotacje do właściwości klasy Contact

- pole niepuste
  @NotNull
- długość tekstu np.`
  @Size(min = 3, max = 30)
- wzorzec np.
  - email - skopiowane z netu
    @Pattern(regexp="\\b[\\w.%-]+@[-.\\w]+\\.[A-Za-z]{2,4}\\b")
  - telefon - proszę użyć google

Dodanie obsługi walidacji do kontrolera ContactController

- walidacja w obsłudze żądania

```
@RequestMapping(value = "/create", method = POST)
public String processContact(@Valid Contact contact, Errors errors) {
if (errors.hasErrors()) {
return "contactForm";
}
contactRepository.insert(contact);
return "redirect:/";
}
```

Pokazanie komunikatów błędu w formularzu contactForm

- dodatkowa kolumna z komunikatem błędu

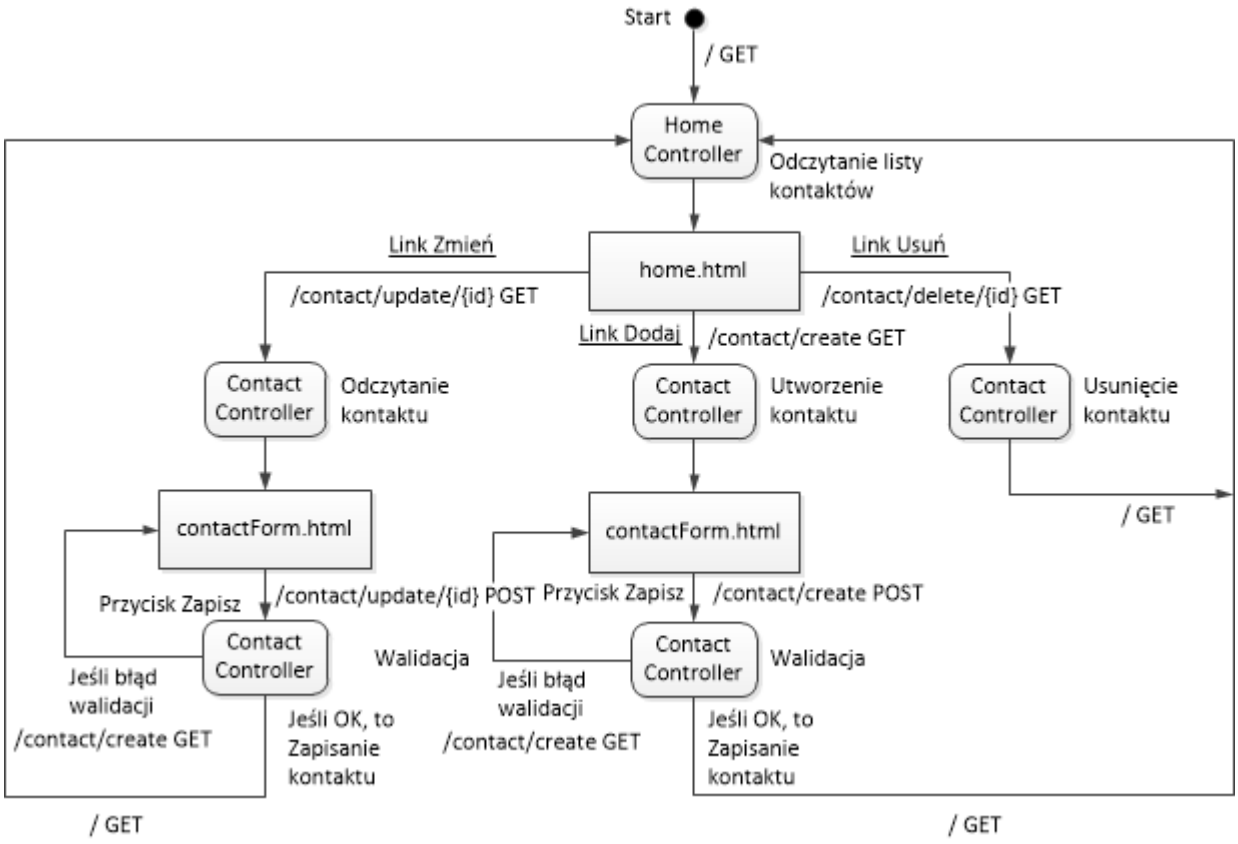
```
<tr>
        <td>Imię:</td>
<td><input type="text" th:field="*{firstName}"/></td>
<td th:errors="*{firstName}"></td>
</tr>
```

Treść zadania

Proszę zaimportować projekt **contacts6**, lub skopiować projekt **contacts5**. Następnie dodać reguły walidacji pól, uruchomić walidację i jej obsługę w kontrolerze oraz zmodyfikować widok, aby wyświetlał komunikaty błedów pól.

Zadanie 7 (nieobowiązkowe)

Celem zadania jest dodanie w aplikacji contacts możliwości zmiany i usuwania kontaktów . Dodatkowa funkcjonalność wymaga rozszerzenia kontrolera kontaktów o obsługę żadań, które powinny być uruchamiane przez linki (dla każdego kontaku osobny link do zmiany i osobny do usunięcia) umieszczone w tabeli pokazującej kontakty w widoku home.html. Docelowy przepływ żądań między widokami i kontrolerami jest pokazany na Rys. 5.4.



Rys 5.4 Przepływ żądań i widoków przy dodawaniu, zmianie i usuwaniu kontaktu z walidacją danych

Rozszerzenie interfejsu ContactRepository

Dla potrzeb kolejnych zadań interfejs repozytorium powinien zawierać dodatkowe metody :

- **findOne** - wyszukanie kontaktu po identyfikatorze
- **update** - zapisanie zmienionego kontaktu

Implementacja interfejsu JdbcContactRepository

Implementacja interfejsu zawiera metody wykorzystujące wstrzyknięty szablon jdbc do wykonywania instrukcji SQL na bazie danych:

- **findOne** - wyszukanie kontaktu po identyfikatorze

```
@Override
public Contact findOne(long id) {
return jdbc.queryForObject(
"SELECT id, first_name, last_name, email, phone" +
" FROM Contact" +
" WHERE id = ?",
new ContactRowMapper(), id);
}
```

- **update** - zapisanie zmienionego kontaktu

```
@Override
public void update(Contact contact) {
jdbc.update("UPDATE Contact "
+ "SET first_name=?, last_name=?, email=?, phone=? "
+ "WHERE id = ?",
contact.getFirstName(),
contact.getLastName(),
contact.getEmail(),
contact.getPhone(),
contact.getId());
}
```

Rozszerzenie kontrolera kontaktów

- żadanie zmiany kontaktu - GET - znajdźcie sami
  @RequestMapping(value = "/update/{id}", method = GET)
- żadanie zmiany kontaktu - POST - znajdźcie sami
  @RequestMapping(value = "/update/{id}", method = POST)
- żadanie usunięcia kontaktu - GET - znajdźcie sami

  …

Linki w widoku home.html

- **Zmień** - znajdźcie sami
- **Usuń** - znajdźcie sami

Treść zadania

Proszę zaimportować projekt **contacts7**, lub skopiować projekt **contacts6**. Następnie rozszerzyć interfejs i implementację repozytorium kontaktów, dodać obsługę żądań zmiany i usuwania do kontrolera kontaktów, umiescić linki do zmiany i usuwania kontaktu w tabeli w widoku **home.html**.