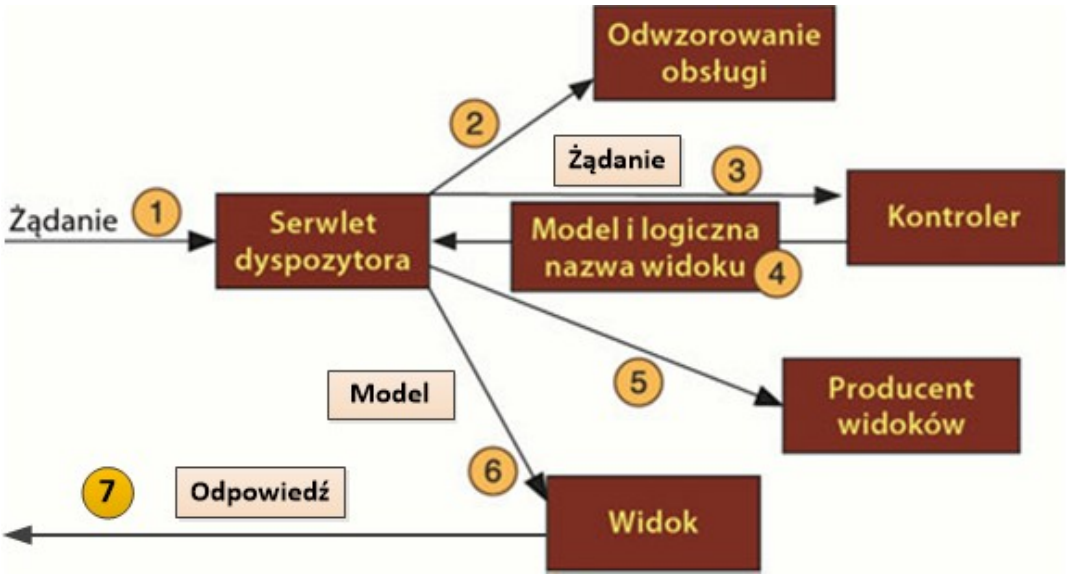
Programowanie komponentowe Spring

Ćw4 - Spring MVC

Wstęp

Laboratorium nr 4 składa się z zadań umożliwiających skonfigurowanie środowiska roboczego i podstawową konfigurację Spring MVC do współpracy z technologią JSP oraz nowszym rozwiązaniem Thymeleaf.
Konfiguracja Spring MVC polega na utworzeniu i połączeniu ze sobą komponentów zapewniających obsługę żądań HTTP wysyłanych przez przeglądarkę klienta.



Rys. 4.1 Obsługa żądania HTTP przez Spring MVC

Proces obsługi żądania został pokazany na Rys. 4.1.

1. Żądanie HTTP z przeglądarki klienta trafia do głównego punktu obsługi - Dyspozytora
2. Dyspozytor na podstawie adresu URL, typu żądania i informacji o odwzorowaniach obsługi określa, który z kontrolerów powinien obsłużyć to żądanie
3. Dyspozytor wysyła żądanie do wybranego kontrolera. Kontroler na podstawie parametrów przesłanych w żądaniu wykonuje zadania (np. pobranie danych z bazy danych)
4. Dane będące wynikiem działania kontrolera są zwracane jako model. Widok prezentujący te dane jest określony przez logiczną nazwę (niezależną od technologii widoków).
5. Dyspozytor używa producenta widoków do przekształcenia logicznej nazwy widoku w konkretną implementację.
6. Dyspozytor przesyła model (dane) do implementacji widoku, który generuje odpowiedź.
7. Wygenerowana odpowiedź jest zwracana do klienta.

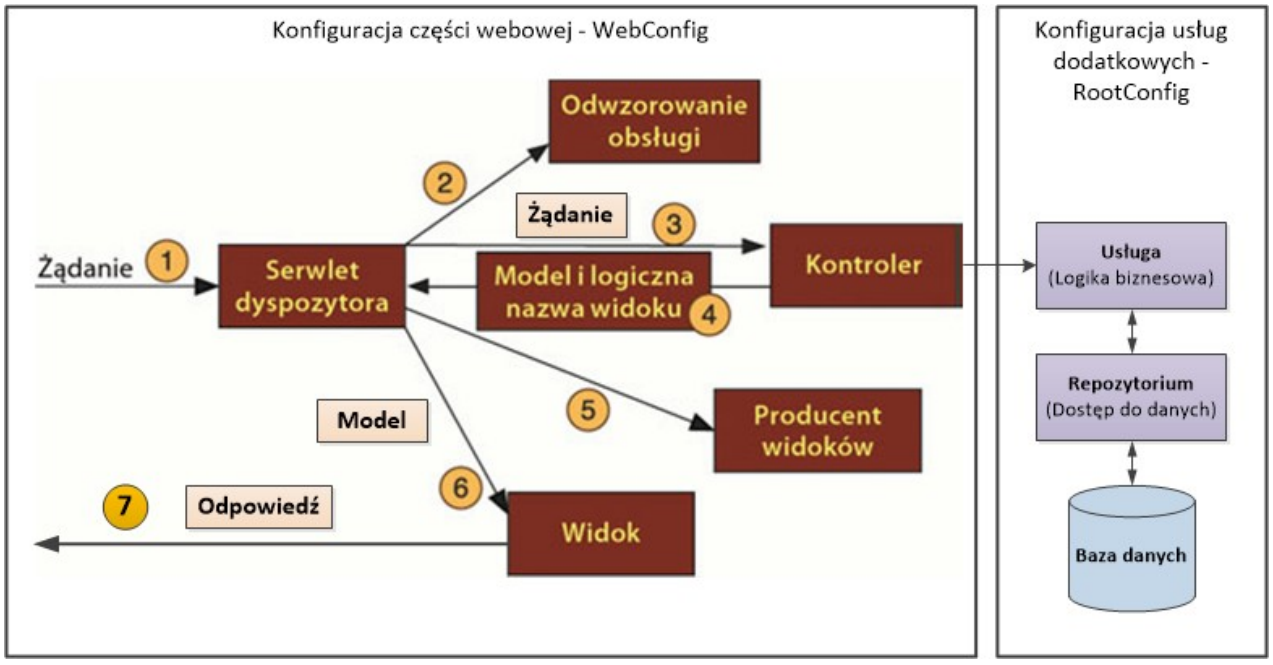
Zadanie 1 - Konfiguracja środowiska

Do wykonania i uruchomienia zadań na laboratorium potrzebny jest lokalny serwer internetowy. Dla tego zastosowania wybrany został serwer Tomcat 9.0, ponieważ jest to technologia sprawdzona, powszechnie używana i dobrze udokumentowana.

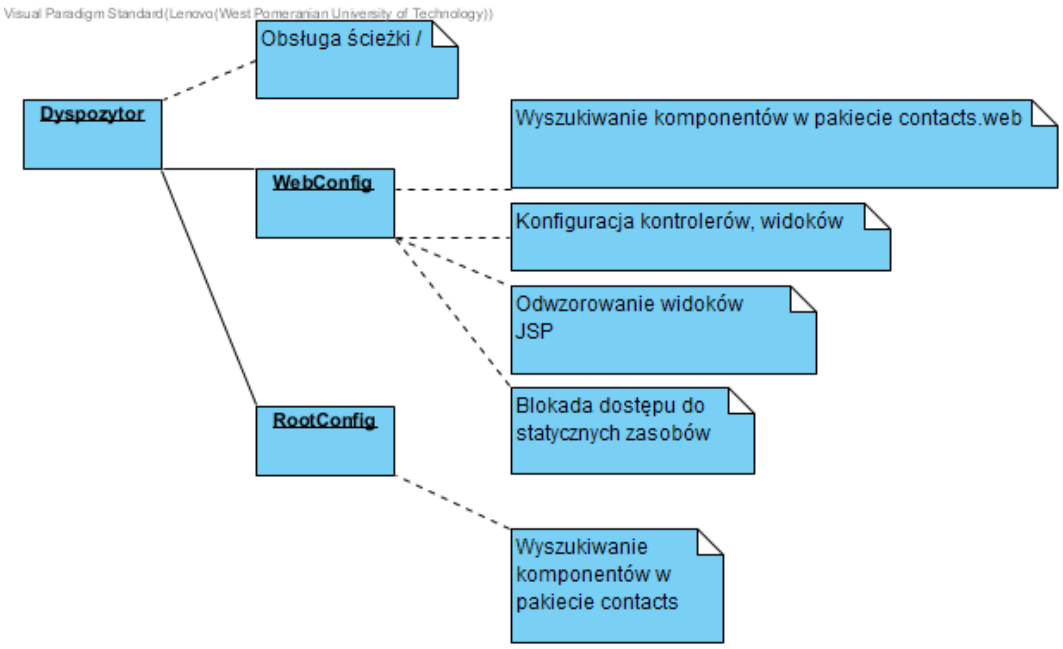
1. Proszę ściągnąć i zainstalować Tomcata 9
    - https://tomcat.apache.org/download-90.cgi
    - 32-bit/64-bit Windows Service Installer
    - uruchomić i sprawdzić, czy działa http://localhost:8080/
    - jeśli nie pojawia się strona startowa Tomcata, zmienić port 8080 na 8888 w pliku *katalog_instalacji*\Tomcat 9.0\conf\server.xml
      <Connector port="8888" protocol="HTTP/1.1"
      connectionTimeout="20000"
      redirectPort="8443" />
    - uruchomić i sprawdzić, czy działa http://localhost:8888/
    - otworzyć okno z serwerami
      Window/Show view/Other
      Server/Servers
      Open
    - usunąć domyślny serwer **Pivotal tc Server v4.0**
    - dodać serwer Tomcat 9 - wskazać na katalog, w którym jest zainstalowany
      Click this link to create new server…

Zadanie 2 - Konfiguracja aplikacji MVC dla technologii JSP

Konfiguracja Spring MVC składa się z dwóch części: konfiguracji części webowej i konfiguracji usług dodatkowych odpowiadających za logikę aplikacji i dostęp do danych, co pokazano na Rys. 4.2.



Rys. 4.2 Dwie konfiguracje Spring MVC



Dyspozytor

W podstawowej konfiguracji Spring MVC główny serwlet dyspozytora powinien rozszerzać klasę *AbstractAnnotationConfigDispatcherServletInitializer* i implementować trzy metody:

- *getServletMappings* - określającą jaką ścieżkę żądania obsługuje dyspozytor, ścieżka "/" powoduje obsługiwanie wszystkich żądań
  protected String[] getServletMappings() {
  return new String[] { "/" };
  }
- *getServletConfigClasses* - zwracającą tablicę klas konfigurujących część webową aplikacji
  protected Class<?>[] getServletConfigClasses() {
  return new Class<?>[] { WebConfig.class }; }
  }
- *getRootConfigClasses* - wskazującą klasy konfigurujące usługi dodatkowe
  protected Class<?>[] getRootConfigClasses() {
  return new Class<?>[] { RootConfig.class };
  }

Konfiguracja części webowej

Klasa konfigurująca część webową powinna rozszerzać klasę *WebMvcConfigurerAdapter* i być opatrzona następującymi adnotacjami:

- **@Configuration** - ponieważ jest to klasa konfiguracjyjna Springa
- **@EnableWebMvc** - w celu włączenia Spring MVC
- **@ComponentScan("contacts.web")** - ponieważ chcemy, aby kontrolery były automatycznie wyszukiwane w pakiecie contacts.web aplikacji.

W klasie konfiguracyjnej należy jeszcze skonfigurować producenta widoków JSP oraz wyłączyć dostęp do statycznych zasobów (plików) aplikacji.

- producent widoków JSP zajmuje się zamianą nazw logicznych widoków na konkretne implementacje w postaci nazw plików w aplikacji
  @Bean
  public ViewResolver viewResolver() {
  InternalResourceViewResolver resolver = new InternalResourceViewResolver();
  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");
  resolver.setExposeContextBeansAsAttributes(true);
  return resolver;
  }
- wyłączenie dostępu do statycznych plików uzyskujemy przez nadpisanie metody
  @Override
  public void configureDefaultServletHandling (
  DefaultServletHandlerConfigurer configurer) {
  configurer.enable();
  }

Konfiguracja usług

Minimalna, podstawowa klasa **RootConfig** konfigurująca usługi dodatkowe powinna zawierać następujące adnotacje:

- @**Configuration** - ponieważ jest to klasa konfiguracjyjna Springa
- @**ComponentScan**(basePackages={"contacts"},
  excludeFilters={
  @Filter(type=FilterType.ANNOTATION, value=EnableWebMvc.class)
  })
  powodująca automatyczne wyszukiwanie w pakiecie *contacts* komponentów springa za wyłączeniem klasy *WebConfig,* co zostało uzyskane przez parametr *excludeFilters*

Kontroler

Kontroler obsługujący żądanie HTTP GET dla głównej strony aplikacji contacts powinien

1. Być opatrzony następującymi adnotacjami:

- @**Controller** - oznaczenie kontrolera Spring MVC
- @**RequestMapping("/")** - określenie ścieżki żądania obsługiwanej przez kontroler
- Zawierać metodę obsługującą żądanie i zwracającą nazwę logiczną widoku
- **public String home(Model model) {**
  return "home";
  }

3. Metoda powinna być opatrzona adnotacją określającą obsługiwaną metodę żądania

- **@RequestMapping(method = GET)**

Treść zadania 2

Proszę zaimportować projekt **contacts1**. Projekt zawiera klasę dyspozytora z pustymi metodami, pustą klasę konfigurującą część webową, pustą klasą konfigurującą usługi dodatkowe oraz pustym kontrolerem. Jedynym działającym elementem aplikacji jest implementacja widoku home.jsp umieszczona w katalogu src/main/webapp/WEB-INF/views.
Proszę na podstawie informacji zamieszconych w skrypcie oraz internecie uzupełnić i połączyć elementy aplikacji a następnie uruchomić ją przez: wciśnięcie prawego przycisku na projekcie / Run As / Run on Server, wybranie serwera Tomcat 9.0, restart.

Zadanie 3 - Konfiguracja aplikacji MVC dla technologii Thymeleaf

Podstawowa konfiguracja Spring MVC do współpracy z widokami Thymeleaf składa się z tych samych elementów, co konfiguracja do współpracy z widokami JSP. Różnica między konfuguracjami występuje tylko w części webowej, która zawiera cztery komponenty zamiast dwóch.

Dyspozytor - bez zmian

Konfiguracja części webowej

Klasa konfigurująca część webową powinna rozszerzać klasę *WebMvcConfigurerAdapter* i być opatrzona następującymi adnotacjami:

- **@Configuration** - ponieważ jest to klasa konfiguracyjna Springa
- **@EnableWebMvc** - w celu włączenia Spring MVC
- **@ComponentScan("contacts.web")** - ponieważ chcemy, aby kontrolery były automatycznie wyszukiwane w pakiecie contacts.web aplikacji.

W klasie konfiguracyjnej należy jeszcze skonfigurować producenta widoków Thymeleaf, silnik szablonów, producenta szablonów oraz wyłączyć dostęp do statycznych zasobów (plików) aplikacji.

- Producent widoków Thymeleaf - tworzy producenta widoków zamieniającego nazwę logiczną widoku na implementację

  ```
  @Bean
  public ViewResolver viewResolver(SpringTemplateEngine templateEngine) {
  ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
  viewResolver.setTemplateEngine(templateEngine);
  return viewResolver;
  }
  ```

- Silnik szablonów - umożliwia parsowanie szablonów i generowanie wyników na podstawie ich zawartości

  ```
  @Bean
  public SpringTemplateEngine templateEngine(TemplateResolver templateResolver) {
  SpringTemplateEngine templateEngine = new SpringTemplateEngine();
  ```

templateEngine.setTemplateResolver(templateResolver);
return templateEngine;
}

- producent szablonów zajmuje się zamianą nazw logicznych szablonów na konkretne implementacje w postaci nazw plików szablonów w aplikacji
  @Bean
  public TemplateResolver templateResolver() {
  TemplateResolver templateResolver = new ServletContextTemplateResolver();
  templateResolver.setPrefix("/WEB-INF/templates/");
  templateResolver.setSuffix(".html");
  templateResolver.setTemplateMode("HTML5");
  return templateResolver;
  }
- wyłączenie dostępu do statycznych plików uzyskujemy przez nadpisanie metody
  @Override
  public void configureDefaultServletHandling (
  DefaultServletHandlerConfigurer configurer) {
  configurer.enable();
  }

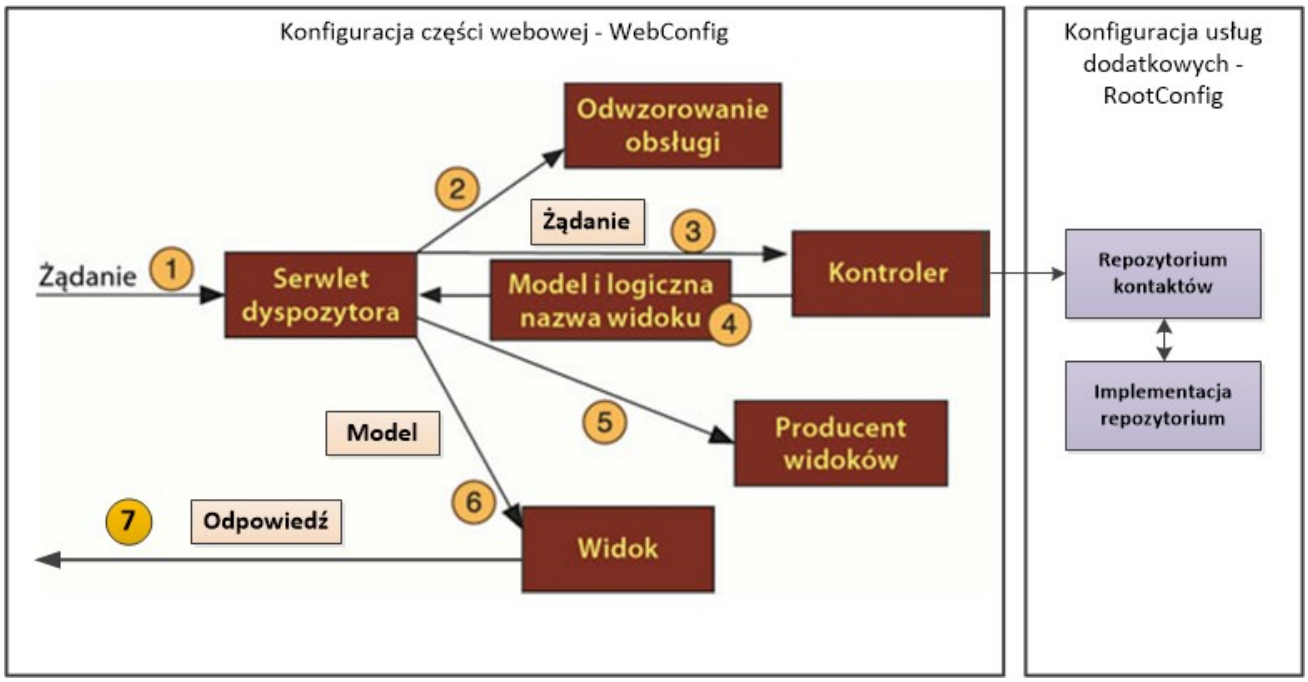
Konfiguracja usług - bez zmian.
Kontroler - bez zmian.
Widok - zrobiony i gotowy.

Treść zadania 3
Proszę zaimportować projekt **contacts2**. Projekt zawiera klasę dyspozytora z pustymi metodami, pustą klasę konfigurującą część webową, pustą klasą konfigurującą usługi dodatkowe oraz pustym kontrolerem. Jedynym działającym elementem aplikacji jest implementacja szablonu home.html umieszczona w katalogu src/main/webapp/WEB-INF/templates.
Proszę na podstawie informacji zamieszconych w skrypcie oraz internecie uzupełnić i połączyć elementy aplikacji a następnie uruchomić ją przez: wciśnięcie prawego przycisku na projekcie / Run As / Run on Server, wybranie serwera Tomcat 9.0, restart.

Zadanie 4 - Konfiguracja aplikacji Spring MVC dla technologii Thymeleaf i warstwy pośredniej

Do aplikacji Spring MVC + Thymeleaf skonfigurowanej w poprzednim zadaniu dołączony zostanie obiekt warswy pośredniej umożliwiający dostęp do repozytorium kontaktów i lista kontaktów z repozytorium zostanie wyświetlona na stronie głównej aplikacji. W tym celu należy utworzyć klasę reprezentującą kontakt oraz interfejs i komponent repozytorium kontaktów. Komponent ten zostanie automatycznie wyszukany dzięki istniejącej konfiguracji usług dodatkowych. Instancja

repozytorium powinna zostać wstrzyknięta do kontrolera strony głównej a zawartość repozytorium - lista kontaktów dołączona do modelu, aby mogła zostać wyswietlona przez szablon home.html.



Rys. 4.3 Warstwa pośrednia - repozytorium kontaktów

Klasa reprezentująca kontakt

Klasa Contact zawiera uproszczone informacje o kontakcie do osoby. Zawiera cztery pola, dwa konstruktory oraz gettery i settery. Kod klasy został dołączony do projektu.

```
public class Contact {
private String firstName;
private String lastName;
private String email;
private String phone;
….
}
```

Komponent repozytorium kontaktów

Komponent składa się z interfejsu i implementacji. Interfejs zawiera jedną metodę pozwalającą na odczytanie listy kontaktów. Implementacja zawiera zakodowaną przykładową listę kontaktów. Zarówno interfejs, jak i implementacja będą rozszerzane i wykorzystywane na następnych zajęciach.

Komponent (implementacja) powinien być oznaczony następującą adnotacją

- **@Component** - aby umożliwić wyszukanie go przez kontener Springa skonfigurowany w RootConfig.java

Kontroler strony głównej

Kontroler strony głównej powinien mieć dostęp do implementacji repozytorium dzięki

- użyciu automatycznego wiązania przez właściwość:

@Autowired
ContactsRepository contactsRepository;

- co pozwoli na dołączenie listy kontaktów do modelu zwracanego do dyspozytora w metodzie home(Model model)
  model.addAttribute("contacts", contactsRepository.findContacts(20, 0));

Widok strony głównej

W widoku strony głównej można wykorzystać znacznik th:each aby wyświetlić na stronie przeglądarki listę przykładowych kontaktów umieszczonych w modelu pod nazwą “contacts”.

- **<tr th:each="contact : ${contacts}">** - dla każdego kontaktu z listy zawartej w modelu, tworzony jest wiersz w tabeli
- odczytanie pól obiektu contact i umieszczenie ich w kolumnach jest możliwe dzięki znacznikowi **th:text**:

```
<td th:text="${contact.firstName}"></td>
<td th:text="${contact.lastName}"></td>
<td th:text="${contact.email}"></td>
<td th:text="${contact.phone}"></td>
```

Treść zadania 4

Proszę zaimportować projekt **contacts3**. Projekt zawiera klasę dyspozytora z pustymi metodami, pustą klasę konfigurującą część webową, pustą klasą konfigurującą usługi dodatkowe oraz pustym kontrolerem. Jedynym działającym elementem aplikacji jest implementacja szablonu home.html umieszczona w katalogu src/main/webapp/WEB-INF/templates.

Proszę na podstawie informacji zamieszconych w skrypcie oraz internecie uzupełnić i połączyć elementy aplikacji, zmienić szablon strony głównej tak, aby wyświetlał listę kontaktów, a następnie uruchomić aplikację na serwerze.