

Ćwiczenia

Lab 1 – Pierwszy Bean

Cel ćwiczenia:

- Uruchomienie Spring
- Stworzenie pierwszego Bean'a

Kroki:

1. W pakiecie pl.altkom.shop.service stworzyć klasę ProductService oraz klasę CoreConfig w pakiecie pl.altkom.shop a w niej metodę która stworzy beana

```
@Bean
public ProductService productService() {
    return new ProductService();
}
```

2. Stworzyć klasę Runner, uruchomić aplikację i pobrać bean'a

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(CoreConfig.class);
ProductService productService = context.getBean("productService");
```

Lab 2 – Wstrzykiwanie

Cel ćwiczenia:

- Wstrzykiwanie zależności

Kroki:

1. Do CoreConfig dodać adnotację @ComponentScan("pl.altkom.shop") która sprawi, że Spring będzie wyszukiwał wszystkich komponentów w danym pakiecie
2. Stworzyć Interfejs ProductRepo oraz jego implementację InMemoryProductRepo
3. Stworzony komponent wstrzyknąć do ProductService za pomocą @Autowired
4. Przetestować różne metody wstrzykiwania pole, seter, konstruktor

Lab 3 – PostProcessor

Cel ćwiczenia:

- Stworzenie specjalnego komponentu Springa

Kroki:

1. Stworzyć klasę która będzie implementowała interfejs BeanPostProcessor
2. Za pomocą Log4j wylogować kolejność tworzenia komponentów

Lab 4 – Cykl życia

Cel ćwiczenia:

- Dodanie produktów podczas utworzenia repozytorium

Kroki:

1. Do repozytorium InMemoryProductRepo dodać mapę przechowującą produkty [id, Product] oraz metodą pobierającą listę wszystkich produktów
2. Dodać metodę inicjalizującą za pomocą której dodać dwa produkty do mapy

Lab 5 – Konfiguracja aplikacji Spring MVC

Cel ćwiczenia:

- Budowa aplikacji web w oparciu o Springa 4.
- Uruchomienie projektu na Tomcat'cie

Kroki:

1. W projekcie znajduje się klasa WebBootstrap przeanalizuj jej zawartość
2. Dodaj projekt do Tomcata 8
3. Wyświetl komunikat podczas dodawania produktów do repo
4. Wyświetl komunikat po uruchomienia aplikacji na serwerze

Lab 6 – Pierwszy kontroler

Cel ćwiczenia:

- Stworzenie kontrolera HelloWorld

Kroki:

1. W pakiecie pl.altkom.shop.controller dodaj kontroler AltkomController
2. Zmapuj kontroler na /altkom
3. Zadeklaruj parametr wejściowy metody jako „HttpServletRequest response” i skorzystaj z niego aby wyświetlić dane dla użytkownika np.
`res.getWriter().append("Witaj na szkoleniu Altkom");`
4. Uruchom kontroler w przeglądarce /shop/altkom

Lab 7 – Kontrolery pobieranie wartości GET

Cel ćwiczenia:

- Odebranie parametrów w kontrolerze

Kroki:

1. Dodaj kontroler ProductController który odczyta wartości parametrów dla /shop/product/list?page=1&size=20&orderBy=name
2. Dodaj mapowanie które odczyta wartość dla mapowania /show/product/12
3. Otrzymane parametry zaprezentuj za pomocą HttpServletResponse response

Lab 8 – Kontrolery przekazywanie danych do widoków

Cel ćwiczenia:

- Prezentacja danych na stronie JSP

Kroki:

1. Informacje z poprzedniego ćwiczenia (/shop/product/list?page=1&size=20&orderBy=name) zaprezentuj na stronie JSP korzystając z wyrażień \${nazwaParametru}. Dodaj odpiedni plik JSP /WEB-INF/pages/product/product-list.jsp(skopiuj index.jsp) oraz zwróć z kontrolera odpowiedni identyfikator widoku który pokaże product/product-list.jsp
2. Pobierz listę wszystkich produktów i wyświetl je w formie tabelki

Lab 9 – Kontrolery reakcja na akcje użytkownika

Cel ćwiczenia:

- Zareagowanie na interakcje użytkownika z aplikacją

Kroki:

1. Dodać kontroler który będzie usuwał produkt po identyfikatorze /product/12/delete – sprawdzić działanie w przeglądarce wpisując adres bezpośrednio
2. Stworzoną funkcjonalność dodać w aplikacji (link)
3. Po usunięciu produktu pokazać odświeżoną listę

Number	Name	Quantity	Price	Actions
1	Rower	12	10	
2	Sanki	123	12.45	
3				

Lab 10 – Kontrolery obsługa dodawania

Cel ćwiczenia:

- Stworzenie formularza dodającego produkt

Kroki:

1. Do product-list.jsp dodaj akcję/linka która pokaże formularz product/new z product-form.jsp
2. Dodaj kontroler obsługujący formularz POST product/new
3. Po zapisie produktu pokaż odświeżoną listę produktów

Lab 11 – Kontrolery edycja

Cel ćwiczenia:

- Stworzenie formularza edytującego produkt

Kroki:

1. Analogicznie jak podczas usuwania produktów dodać akcję edytującą dany produkt
2. W kontrolerze pobrać produkt o id i wyświetlić dane na formularzu (product-form)
3. Podczas zapisu zaktualizować wpis o produkcie i przejść do listy

Lab 12– Lokalizacja

Cel ćwiczenia:

- Wielojęzyczność w aplikacji

Kroki:

1. Skonfigurować ResourceBundleMessageSource
2. Założyć plik messages.properties
3. W aplikacji zamienić wszystkie stringi tak aby korzystały z `<spring:message code="name.label"/>`
4. Dodać plik messages_en.properties i sprawdzić działanie w sytuacji kiedy zmienimy locale w przeglądarce

Lab 13 – Walidacja formularzy

Cel ćwiczenia:

- Walidacja formularzy

Kroki:

1. W kontrolerze wstrzyknąć jako parametr metody obiekt BindingResult
2. Dodać błąd dla pola reject
3. Wyświetlić błąd na formularzu

Lab 14 – Walidacja formularzy deklaratorywnie

Cel ćwiczenia:

- Walidacja formularzy JSR 303

Kroki:

1. Dodać zależność maven hibernate-validator
2. Dodać adnotacje NotNull/NotEmpty na odpowiednie pola
3. Dodać adnotację @Valid do parametru wejściowego
@ModelAttribute @Valid Product product
4. Wyświetlić błędy na formularzu

Lab 15 – Widok PDF

Cel ćwiczenia:

- Praca z innymi typami widoków

Kroki:

1. Skonfigurować BeanNameViewResolver
2. Dodać bean ProductListPDFView który dziedziczy po AbstractPdfView i korzystając z api Table stworzyć tabelkę

```
Table table = new Table( 1 );  
table.addCell(„Name”)  
table.addCell(„Wartość”)  
document.add(table)
```

Lab 16 – Kontrolery REST

Cel ćwiczenia:

- Tworzenie kontrolerów Rest'owych
- Testowanie za pomocą Soap-UI

Kroki:

1. Dodaj kontroler RestPhoneController z mapowaniem /rest/product
2. Korzystając z adnotacji @RestController oraz @RequestBody zaimplementuj wszystkie funkcjonalności kontrolera PhoneController
3. Zaobserwuj jak działa walidacja
4. Dodaj funkcjonalność wyszukiwania produktów po nazwie

Lab 17 – Swagger

Cel ćwiczenia:

- Dodać bibliotekę Swagger

Kroki:

1. <http://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>
2. Zależności
3. Config
4. Mapowanie resource'ów

Lab 18 – Hibernate

Cel ćwiczenia:

- Stworzenie HibernateProductRepository który będzie implementował interfejs ProductRepository

Kroki:

1. Przeanalizować zawartość pakietu pl.altkom.shop.model
2. Przeanalizować konfigurację pl.altkom.shop.DBConfig
3. Zaimplementować operację z ProductRepository za pomocą Hibernate'a
4. Sprawdzić działanie za pomocą aplikacji

Lab 20 – Hibernate - zapytania

Cel ćwiczenia:

- Wytworzenie w SaleDocumentRepo metod umożliwiających zaawansowane wyszukiwanie dokumentów

Metody:

1. Wyszukiwanie dokumentu po numerze
2. Wyszukiwanie dokumentów przekraczających kwotę z parametru
3. Zwrócenie dokumentów posortowanych po kwocie malejąco
4. Zwrócenie dokumentów posiadających co najmniej 3 pozycje
5. Zwrócenie 10 pierwszych dokumentów
6. Zwrócenie x pierwszych dokumentów pomijając y wierszy gdzie x,y parametry
7. Pobranie dokumentu odrazu z wszystkimi pozycjami

Lab 21 – AOP

Cel ćwiczenia:

- Stworzenie prostego aspektu do monitorowania aplikacji

Metody:

1. Dodaj adnotację Monitoring
2. Dodaj Aspect który będzie tropił wszystkie wwołania metod oznaczonych adnotacją Monitoring i wypisywał je na konsole za pomocą log4j

```
@Component
```

```
@Aspect
```

```
public class MonitroingAspect {
```

```
    @Around("@annotation(pl.altkom.shop.aop.Monitoring)")
```

```
    public Object monitpr(ProceedingJoinPoint pjp) throws Throwable {
```

```
        Object obj = pjp.proceed();
```

```
        return obj;
```

```
    }
```

```
}
```

Lab 22 – AOP - rozbudowa

Cel ćwiczenia:

- Dodanie parametru do aspektu Monitoring mówiącego o maksymalnym czasie wykonania metody. W razie przekroczenia logować na konsoli na poziomie ERROR

Metody:

1. Dodaj parametr do adnotacji Monitoring

Lab 23 – AOP - cache

Cel ćwiczenia:

- Dodać adnotację `ResponseCache` która będzie cache'ować wyniki wywołania metod oznaczonych adnotacją

Metody:

1. Dodaj adnotację `ResponseCache`
2. Dodać aspekt cache'ujący

Lab 24 – Transakcje

Cel ćwiczenia:

- Skorzystać z adnotacji @Transactional w odpowiednich miejscach

Metody:

1. Na poziomie serwisów dodać adnotację @Transactional
2. W repo skorzystać(wstrzyknąć) z

@PersistenceContext

EntityManager em;

3. Sprawdzić działanie w SaleDocumentService czy wyjątek wycofuje wszystkie zmiany
4. Jak obsługiwać wyjątki w transakcji/ korzystać z try/catch?

Lab 25 – Transakcje readOnly

Cel ćwiczenia:

- Skorzystać z adnotacji `@Transactional(readOnly)` dla metod odczytujących

Kroki:

1. Na poziomie metod odczytujących dodać adnotację `@Transactional(readOnly=true)`
2. Co się stanie jeżeli będziemy chcieli zapisać coś w takiej metodzie?
3. Co się stanie jeżeli uruchomimy metodę modyfikującą z serwisu i w trakcie zwołamy metodą tylko odczytującą?

Lab 26 – Bezpieczeństwo wywołań metod

Cel ćwiczenia:

- Zapoznać się z podstawową konfiguracją Spring-security
- Umożliwić wywołanie funkcji dodaj produkt tylko użytkownikom z rolą ADMIN

Kroki:

1. Dodać do metody insert serwisu lub repozytorium od produktów odpowiednią adnotację `@Secured("ROLE_ADMIN")`
2. Sprawdzić działanie aplikacji podczas logowania się na użytkownika z rolą USER oraz ADMIN

Lab 27 – Bezpieczeństwo WWW – własna strona logowania

Cel ćwiczenia:

- Dodać możliwość logowania za pomocą specjalnie przygotowanej strony login.jsp

Kroki:

1. W sekcji `<http>` dodać `<form-login login-page="/login" />`
2. Sprawdzić działanie nowej strony logowania

Lab 28 – Bezpieczeństwo WWW – webservice

Cel ćwiczenia:

- Dodać zabezpieczenie dla webserviceów aby logowały się do systemu za pomocą BasicAuthentication

Kroki:

1. Dodać nowy wpis w pliku security.xml który dla wzroca /api/** użyje logowania za pomocą <basic-auth/>

Lab 29 – Bezpieczeństwo WWW – ukrywanie elementów

Cel ćwiczenia:

- Pokazywać akcję „Dodaj nowy produkt” tylko dla użytkowników z rolą ADMIN

Kroki:

1. Skorzystać z taga `<sec:authorize access="hasAnyRole('ROLE_ADMIN')">` wewnątrz którego umieścić akcję dodającą. Jeżeli użytkownik będzie miał odpowiednią rolę wewnątrz taga zostanie wyrenderowane

Lab 30 – WebService CXF

Cel ćwiczenia:

- Analiza konfiguracji CXF

Kroki:

1. Przeanalizować zmiany w pliku WebBootstrap
2. Przeanalizować plik CXFConfig
3. Omówić idee działania – interceptorów/feature'ów

Lab 31 – WebService CXF - REST

Cel ćwiczenia:

- Stworzyć webservice typu REST za pomocą CXF korzystając ze standardu JAX-RS

Kroki:

1. W pakiecie pl.altkom.shop.cxf założyć klasę ProductJAXRSWebService
2. Za pomocą adnotacji @Path, @GET, @POST stworzyć metody pobierające listę produktów oraz metodą umożliwiającą dodanie nowego produktu
3. Sprawdzić wygenerowany dokument WADL
<http://localhost:8080/shop/services/>

przykład:

@Component

@Path("/saleDocument")

@Produces(MediaType.APPLICATION_JSON)

```
public class SaleDocumentRESTWebService {  
    @GET  
    @Path("/{id}")  
    public SaleDocument findById(@PathParam("id") Long id) {  
        SaleDocument saleDocument = new SaleDocument();  
        saleDocument.setNumber("REST " + id);  
        return saleDocument;  
    }  
}
```

Lab 32 – WebService CXF - SOAP

Cel ćwiczenia:

- Stworzyć webservice typu SOAP za pomocą CXF korzystając ze standardu JAX-WS

Kroki:

1. W pakiecie pl.altkom.shop.cxf założyć klasę ProductJAXWSWebService
2. Za pomocą adnotacji @WebService, @WebMethod stworzyć metody pobierające listę produktów oraz metodą umożliwiającą dodanie nowego produktu
3. Sprawdzić wygenerowany dokument WSDL
<http://localhost:8080/shop/services/>

przykład:

@Component

@WebService

```
public class SaleDocumentSOAPWebService {  
    @WebMethod  
    public SaleDocument findById(Long id) {  
        SaleDocument saleDocument = new SaleDocument();  
        saleDocument.setNumber("SOAP " + id);  
        return saleDocument;  
    }  
}
```

Lab 33 – WebService JAX-WS – SOAP client

Cel ćwiczenia:

- Wygenerowanie klienta dla webservice typu SOAP

Kroki:

1. Przechodzimy do katalogu JDK Java (C:\Program Files (x86)\Java\jdk1.8.0_101\bin)
2. Uruchamiamy konsolę
3. Wywołujemy polecenie `wspimport -p pl.altkom.shop.wsclient.soap http://localhost?wsdl`
gdzie `-p` to nazwa pakietu wygenerowanych klas, a koniec polecenia to adres usługi WSDL dla której chcemy wygenerować klient
4. W katalogu bin utworzy się katalog pl który zawiera nasze źródła (pliki java)
5. Wygenerowane źródła przekopiuujemy do eclipse do odpowiedniego pakietu
6. Wywołujemy wygenerowany kod

przykład:

```
JAXWSService JAXWSService = new JAXWSService();  
    SaleDocumentSOAPWebService saleDocumentSOAPWebServicePort =  
JAXWSService.getSaleDocumentSOAPWebServicePort();  
SaleDocument findById = saleDocumentSOAPWebServicePort.findById(1L);
```