

# Spring MVC ćwiczenia

- Spring 4.2
- Spring security 4.1
- Hibernate 5 jako provider JPA
- Java 8
- Maven 3
- Tomcat 8

*Celem ćwiczeń jest stworzenie prostej aplikacji która będzie obsługiwała aukcje. Wraz z postępem szkolenia do naszej aplikacji będziemy dodawali kolejne elementy takie jak: kontrolery, widoki, aspekty, JPA, transakcje, bezpieczeństwo.*

*Poniższe ćwiczenia będziemy wykonywać w obrębie projektu spring-core. W razie problemów można się wspomagać projektem spring-core-solved który zawiera już gotowe rozwiązania.*

## 1.0: Rozpoczynamy od analizy aktualnych zależności maven'a

### 1.1: Stworzenie podstawowej konfiguracji Spring i stworzenie pierwszego komponentu/bean'a

**Zadanie:** W pakiecie `pl.vavatech.auction.blc.service` należy stworzyć klasę `UserService` a w pakiecie `pl.vavatech.auction.blc` klasę konfiguracji.

```
@Configuration //informacja dla springa, że to jest klasa konfiguracyjna
public class BusinessConfig {
    @Bean
    UserService userService() {
        return new UserService();
    }
}
```

Konfigurację uruchamiamy za pomocą:

```
ApplicationContext context = new AnnotationConfigApplicationContext(BusinessConfig.class);
```

Należy sprawdzić jakie metody udostępnia obiekt context i pobrać bean'a userService.

## 1.2: Automatyczne wykrywanie komponentów

**Zadanie:** W pakiecie *pl.vavatech.auction.blc.service* należy utworzyć klasy serwisów AuctionService, OfferService a w pakiecie *pl.vavatech.auction.blc.repo* klasy repozytoriów AuctionRepo, OfferRepo.

Klasy serwisów oznaczamy adnotacją `@Service` a klasy repozytoriów adnotacją `@Repository`.

W klasie BusinessConfig dodajemy adnotację `@ComponentScan("pl.vavatech.auction.blc")` która oznacza, że spring ma szukać komponentów we wskazanym pakiecie (i podpakietach).

Należy sprawdzić za pomocą obiektu context czy spring wyszukał wszystkie komponenty.

**Dokumentacja:**

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-scanning-autodetection>

## 1.3: Poznanie cyklu życia komponentów

**Zadanie:** W pakiecie [pl.vavatech.auction.blc.model](#) znajdują się encje. Zaimplementować obiekty `InMemoryAuctionRepo`, `InMemoryOfferRepo` dodając im metody:

```
Entity find(Long id)
List<Entity> findAll()
void update(entity)
Long insert(entity)
void delete(Long id)
```

Aktualna implementacja może opierać się o mapę w pamięci. Obiekty `InMemory` są implementacjami interfejsów `AuctionRepo` oraz `OfferRepo` (refaktor class → interface).

Za pomocą metod cyklu życia podczas inicjalizacji komponentów `InMemory` dodać dwie aukcje.

**Dokumentacja:**

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-java-lifecycle-callbacks>

## 1.4: Stworzenie procesora beanów

**Zadanie:** W pakiecie *pl.vavatech.auction.blc.component* tworzymy klasę BeanProcessor implementującą BeanPostProcessor. Za pomocą logera(Log4J) wypisać kolejność inicjalizacji komponentów.

**Dokumentacja:**

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#context-introduction-ctx-vs-beanfactory>

## 1.5: Skorzystanie z plików konfiguracyjnych

**Zadanie:** W resources tworzymy plik *application.properties* a w nim klucz *maxShippingPrice=1000*. Wartość klucza należy pobrać w serwisie AuctionService za pomocą adnotacji `@Value`. Aby plik został wczytany należy skorzystać

```
@PropertySource(value = { "classpath:application.properties" })
```

w klasie konfiguracji oraz zdefiniować

```
@Bean
public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
    return new PropertySourcesPlaceholderConfigurer();
}
```

**Dokumentacja:** [http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#\\_\\_propertysource](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#__propertysource)

## 1.6: Wstrzykiwanie zależności

**Zadanie:** Do komponentów `AuctionService` oraz `OfferService` wstrzyknąć odpowiadające im repozytoria. Wypróbować wszystkie możliwości wstrzyknięć: constructor, setter, field.

**Dokumentacja:**

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#beans-autowired-annotation>

## 1.7: Skorzystanie ze wstrzykniętego obiektu

**Zadanie:** W pakiecie `pl.vavatech.auction.blc.service` w klasie `AuctionService` dodać metody, który delegują wywołania do `AuctionRepo`:

`Entity find(Long id)`

`List<Entity> findAll()`

`void update(entity)`

`Long insert(entity)`

`void delete(Long id)`

Sprawdzić działanie, dodać aukcje.



## 1.8: Backdoor za pomocą SpringEL

**Zadanie:** W pakiecie *pl.vavatech.auction.blc.component* stworzyć komponent BackDoor, który będzie posiadał metodę przyjmującą wyrażenie **SpringEL** i zwracał wynik jako object. Należy zdobyć referencję do ApplicationContext i ustawić go jako BeanResolvera dla StandardEvaluationContext. Po stworzeniu komponentu pobrać dowolną aukcję za pomocą AuctionRepo.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#expressions>

*Poniższe ćwiczenia będziemy wykonywać w obrębie projektu spring-mvc. W razie problemów można się wspomagać projektem spring-mvc-solved który zawiera już gotowe rozwiązania.*

*Od tej chwili wszystkie projekty będą posiadały warstwę webową do której potrzebujemy kontenera servletów w naszym przypadku należy skorzystać z Tomcat 8.*

## 2.0: Rozpoczynamy od analizy aktualnych zależności maven'a

### 2.1: Przeanalizować konfigurację Spring MVC w oparciu o javaConfig klasy **AppInitializer** oraz **AppConfig**.

### 2.2: Dodanie pierwszego kontrolera

**Zadanie:** W pakiecie `pl.vavatech.auction.www.controller` stworzyć kontroler `HelloWorldController(/helloWorld)` który pokaże przygotowany widok `helloWorld.jsp` z wartością modelu `message`.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-controller>

## 2.3: Odczytanie parametrów ze ścieżki oraz żądania

**Zadanie:** W kontrolerze `HelloWorldController` dodać mapowanie aby obsługiwał linki typu `/helloWorld/month/type` gdzie month będzie od 1-12 a type to dowolny String.

Otrzymane wartości wyświetlić w `helloWorld.jsp`. Dodatkowo w kontrolerze odebrać parametry żądania ?  
`page=1&pageSize=20` i również wyświetlić na stronie.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-ann-methods>

## 2.4: Odczytanie danych formularza przesłanych przez Postman

**Zadanie:** W kontrolerze `HelloWorldController` dodać mapowanie `/hellWorld/add` tak aby jego parametrem wejściowym był Auction i za pomocą Postman(chrome) wysłać żądanie x-www-form które zawierające title oraz description.

## 2.5: Walidacja otrzymanych danych

**Zadanie:** W kontrolerze `HelloWorldController` w mapowaniu `/hellWorld/add` dodać adnotację `@Valid` do parametru metody. I za pomocą postman wysłać błędne żądanie np. wartość enuma `auctionType` która nie istnieje. Sprawdzić co się stanie.

Dodać parametr metody `BindingResult` który zawiera błędy walidacyjne. Jeżeli obiekt `BindingResult` będzie zawierał błędy to wyrzucić wyjątek np. `IllegalArgumentException`.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#validation>

## 2.6: Interceptor logujący czas obsługi żądania

**Zadanie:** W pakiecie *pl.vavatech.auction.www.component* dodać interceptor `RenderingTimeInterceptor` który będzie logował na konsoli czas w ms który był potrzebny na obsługę żądania.

Interceptor zarejestrować w AppConfig metoda `addInterceptors`.

## 2.7: Obsługa błędów

**Zadanie:** W kontrolerze `HelloWorld` dodać obsługę wyjątków za pomocą `@ExceptionHandler`'a

która przekieruje na stronę `'cmm/error'`.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-exceptionhandlers>

## 2.8: Globalna obsługa błędów

**Zadanie:** W kontrolerze [HelloWorld](#) usuwamy obsługę błędów i dodajemy [SimpleMappingExceptionHandler](#) który w sytuacji wystąpienia błędu w dowolnym kontrolerze przekieruje na stronę '[cmm/error](#)'.

## 2.9: Pokazanie wszystkich aukcji w systemie

**Zadanie:** W pakiecie `pl.vavatech.auction.www.controller.auction` znajduje się `AuctionController` który będzie obsługiwać `/auctions` gdzie w wyniku zostanie pokazany widok `auciton/list` wraz ze wszystkimi aukcjami w systemie. Akcje należy pobrać za pomocą serwisu `AuctionService`

W przeanalizować plik `auciton/list.jsp` oraz dodać kolumnę „Tytuł”.

## 2.10: Pokazanie szczegółów wybranej aukcji

**Zadanie:** W `AuctionController` dodać obsługę `/auctions/idAukcji/show` gdzie kontroler pobierze szczegóły wybranej aukcji i pokaże widok `auction/show`.

W pliku `auciton/show.jsp` należy pokazać wszystkie dostępne pola.

Dodać obsługę przejścia(link) do szczegółów aukcji z poziomu list aukcji

## 2.11: Obsługa edycji

**Zadanie:** W [AuctionController](#) znajduje się obsługa [auctions/id/edit](#) oraz [auctions/save](#). Przeanalizować zasadę działania obsługi formularza. Jak wchodzi w tryb edycji oraz co się dzieje po zapisz.

## 2.12: Obsługa usuwania

**Zadanie:** W [AuctionController](#) dodać [auctions/id/delete](#) która usunie wybrany rekord oraz dodać obsługę usuwania z poziomu listy.

## 2.13: Walidator dla aukcji

**Zadanie:** Napisać i wykorzystać walidator dla aukcji który sprawdzi czy data ważności aukcji nie jest przeszła.

Dodatkowo skonfigurować walidator JSR 303 (dodać zależność maven'a hibernate-validator)

i dodać wymagalność pól (title, description) w klasie aukcji. Dodatkowo dla pola title zdefiniować maksymalną ilość znaków na 20.

Sprawdzić działanie zdefiniowanych ograniczeń na formularzu.



## 2.14: Formater

**Zadanie:** Napisać formater (AnnotationFormatterFactory) który dla stworzonej adnotacji `CurrencyFormat` będzie dodawał PLN do wartości liczbowych. Skorzystać z adnotacji `CurrencyFormat` w klasie Auction w polu `shippingPrice`.

## 2.15: Kontrolery REST (testujemy Postman'em)

**Zadanie:** Stworzyć kontroler `AuctionRestController (@RestController)` który umożliwia operacje `CRUD /rest/auctions`.

Należy rozpocząć od operacji `findAll` i sprawdzić jak wygląda list aukcji w postaci `JSON`.

Podczas implementacji POST'a skorzystać z adnotacji `@RequestBody` dla parametru metody Auction aby spring wiedział, że cały request JSON powinien mapować na ten właśnie obiekt.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#mvc-ann-restcontroller>

## 2.16: Kontrolery REST – obsługa błędów

**Zadanie:** Sprawdzić co się stanie jeśli do naszego controlera REST wyślemy niepoprawne dane skorzystać z adnotacji `@Valid`.

Co się stanie jeżeli podczas przetwarzania logiki poleci błąd ?

## 2.17: Kontrolery REST – struktury danych

**Zadanie:** Stworzyć kontroler `OfferRestController (/rest/offers)`, który umożliwi składanie ofert do trwających aukcji.

Zastanowić się jaka powinna być struktura obiektu który będzie za to odpowiadał.

Dodać regułę walidacyjną która stwierdzi czy proponowana oferta nie jest niższa od aktualnej.

Algorytm powinien odnaleźć aukcję której dotyczy oferta ustawić jej aktualną cenę z oferty następnie dodać ofertę do systemu.

## 2.18: [Opcjonalne] Integracja ze Swagger

### Zależności:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.5.0</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.5.0</version>
</dependency>
```

### Mapowanie:

```
registry.addResourceHandler("swagger-ui.html").addResourceLocations("classpath:/META-INF/resources/");
registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/resources/webjars/");
```

### Konfiguracja:

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Bean
    public Docket api() {
        Predicate<RequestHandler> handlersToInclude = RequestHandlerSelectors.any();
        Predicate<String> pathToInclude = PathSelectors.regex("/rest/.*");

        return new Docket(DocumentationType.SWAGGER_2).select().apis(handlersToInclude)
            .paths(pathToInclude).build();
    }
}
```

*Poniższe ćwiczenia będziemy wykonywać w obrębie projektu spring-ajt. W razie problemów można się wspomagać projektem spring-ajt-solved który zawiera już gotowe rozwiązania*

## 3.0: Rozpoczynamy od analizy aktualnych zależności maven'a

### 3.1: AOP – podstawy

**Zadanie:** Należy przeanalizować działanie klasy [pl.vavatech.auction.blc.aop.sample.JavaMagic](#) która jest przykładem programistycznego wykorzystania API Springa do kreowania Proxy. Jest to podstawa wszystkich magicznych mechanizmów w Springu.

## 3.2: AOP - monitor

**Konfiguracja:** Oprócz dodanych zależności do prawidłowego działania potrzebujemy skorzystać z adnotacji `@EnableAspectJAutoProxy` w klasie `BusinessConfig`.

**Zadanie:** W pakiecie `pl.vavatech.auction.blc.aop` dodać adnotację `@Trace`, którą będziemy umieszczać na metodach serwisu które będziemy chcieli obserwować pod kątem wydajności. Dodać odpowiedni `Aspect` który wypisze na konsole czas potrzebny na wykonanie metody.

<https://blog.espenberntsen.net/2010/03/20/aspectj-cheat-sheet/>

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#aop-pointcuts-designators>

## 3.3: AOP - parametry

**Zadanie:** Do adnotacji `@Trace` dodajemy parametr mówiący o maksymalnym czasie wykonania metody w razie przekroczenia tego czasu należy zalogować taką informację na poziomie ERROR dodatkowo wypisując parametry metody dla których to wystąpiło.

Dokumentacja : <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#aop>

## 3.4: JPA

**Zadanie:** Odkomentować i przeanalizować konfigurację JPA oraz TX w pliku BusinessConfig

Stworzyć nowe implementacje dla AuctionRepo oraz OfferRepo które będą zapisywały obiekty do bazy danych za pomocą JPA `@PersistenceContext EntityManager`.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-jpa>

## 3.5: TX

**Konfiguracja:** Oprócz dodanych zależności do prawidłowego działania potrzebujemy skorzystać z adnotacji `@EnableTransactionManagement` w klasie BusinessConfig.

**Zadanie:** Aktualnie dodawanie aukcji oraz ofert nie działa bo system informuje o braku transakcji aby to naprawić należy dodać adnotację `@Transactional` w miejscach w których chcemy aby nasze metody/klasy były transakcyjne.

Dokumentacja: <http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#orm-jpa>



## 3.6: JPA zapytania

**Zadanie:** W kontrolerze restowym od aukcji dodać możliwość pobierania danych o aukcjach. Wymagania:

- Wyszukiwanie po tytule i opisie
- Sortowanie po dacie lub cenie
- Stronnicowanie wyników, rozmiar strony

Skorzystać z atrybutu `readOnly` w adnotacji `@Transactional`

## 3.7: [Opcjonalne] Testy

**Zadanie:** W projekcie znajdują się klasy `AuctionServiceTest`, `OfferServiceTest` oraz `AuctionRepoTest`.

Dodać testy sprawdzające poprawność funkcji znajdujących się w odpowiadających im serwisach/repozytoriach

*Poniższe ćwiczenia będziemy wykonywać w obrębie projektu spring-security.*

*W razie problemów można się wspomagać projektem spring-security-solved który zawiera już gotowe rozwiązania.*

## 4.0: Rozpoczynamy od analizy aktualnych zależności maven'a

### 4.1: Przeanalizować konfigurację Spring Security

Klasę `pl.vavatech.auction.AppInitializer`, filtr `springSecurityFilterChain`

oraz `pl.vavatech.auction.www.SecurityConfig` i plik `spring-security.xml` na który wskazuje powyższa klasa.

## 4.2: Web Security

**Zadanie:** W pliku `spring-security.xml` należy zdefiniować reguły bezpieczeństwa zabraniające przeglądania aukcji (/auctions) niezalogowanym użytkownikom. Logowanie za pomocą LoginForm  
Dodatkowo tylko użytkownicy z rolą ADMIN mogą dodawać i usuwać aukcje.

W plikach `nav.jsp` oraz `list.jsp` zostały dodane odpowiednie sekcje ukrywające zawartość w zależności od roli użytkownika

```
<sec:authorize access="hasRole('ADMIN')">
    <a href="/auctions/{auction.id}/delete">
        <i class="glyphicon glyphicon-remove-circle"></i>
    </a>
</sec:authorize>
```

Dokumentacja : <http://docs.spring.io/spring-security/site/docs/4.1.1.RELEASE/reference/htmlsingle/#ns-minimal>

<http://docs.spring.io/spring-security/site/docs/4.1.1.RELEASE/reference/htmlsingle/#overview>

## 4.3: Web Security

**Zadanie:** W pliku `spring-security.xml` należy zdefiniować reguły bezpieczeństwa pozwalające korzystania z webservice'ów tylko dla użytkowników z rolą „REMOTE”. Logowanie w webservice powinno się odbywać za pomocą Basic Authorization

<http://docs.spring.io/spring-security/site/docs/4.1.1.RELEASE/reference/htmlsingle/#overview>

## 4.4: Method Security

**Zadanie:** W aplikacji tylko użytkownik z rolą ADMIN może dodać Aukcje jednak jest to tylko i wyłącznie ukrycie linka. Zabezpieczyć metodę serwisu AuctionService tak aby dodać aukcję mógł tylko użytkownik z rolą ADMIN. Skorzystać z adnotacji `@PreAuthorize`.

Dokumentacja: <http://docs.spring.io/spring-security/site/docs/3.2.9.RELEASE/reference/htmlsingle/#ns-method-security>

## 4.5: Method Security expressions

**Zadanie:** Zabezpieczyć AuctionService update tak aby użytkownicy mogli edytować tylko swoje aukcje.

Na początek musimy tak zmodyfikować serwis od dodawania aby zapisał do pola creatorUserName login aktualnie zalogowanego użytkownika. Następnie skorzystać z adnotacji `@PreAuthorize`.