

# **SPRING i SPRING BOOT**

**Sylvia Oleś i Arkadiusz Kasprzak**



**AGH**

**Akademia Górniczo-Hutnicza  
Im. Stanisława Staszica w  
Krakowie**

# Plan prezentacji

1. Wprowadzenie do Spring Boot.
2. Działanie Spring Boot.
3. Spring Boot Actuator.
4. Logowanie zdarzeń - biblioteka SLF4J.
5. Spring Security.
6. Testowanie aplikacji w Springu.

# Materiały


- Link: <https://github.com/arokasprz100/Spring-seminars>
- Repozytorium zawiera:
  - prezentacje
  - przykłady
  - zadanie domowe wraz z instrukcją




1

# **Wprowadzenie do Spring Boot**


# Po co używać Spring Boot?

- 
- Używanie frameworka Spring często niesie za sobą konieczność długiej i powtarzalnej konfiguracji używanych w projekcie zależności.
  - Konieczne jest tworzenie sporej liczby plików .xml i/lub klas, często na zasadzie kopiowania gotowych rozwiązań
  - Takie podejście daje z jednej strony dużą elastyczność,  
■ ale jeśli chcemy po prostu zrobić coś w sposób standardowych to dodaje nam sporo pracy.
  - Spring Boot jest odpowiedzią na ten problem.

# Co to właściwie jest Spring Boot?


- 
- Zbiór predefiniowanych konfiguracji pozwalających w prosty sposób korzystać z domyślnych rozwiązań - może to być np. dodanie do aplikacji obsługi jakiejś bazy danych.
  - Bardzo prosta zasada działania: na etapie **startowania aplikacji** Spring Boot skanuje *classpath* (lokalizacja, w której znajdują się pliki *.class* i pakiety) i na podstawie jego zawartości **konfiguruje te komponenty**, które są nam potrzebne.
  - W dalszej części prezentacji pokazane zostaną niektóre szczegóły działania tego mechanizmu.

# Co to właściwie jest Spring Boot?

- 
- Ponadto Spring Boot:
    - dostarcza narzędzi do monitorowania stanu aplikacji.
    - dostarcza narzędzia wzbogacające możliwość pisania testów.
    - dostarcza wbudowany serwer Tomcat (i możliwość zamiany na Jetty czy Undertow).

Spring Boot nie jest więc jedynie narzędziem do szybkiego generowania projektów.

# Spring vs. Spring Boot

- 
- **Przykład 1:** Porównanie aplikacji napisanej w Spring z aplikacją w Spring Boot.
    - prosta aplikacja pozwalająca na tworzenie i wyświetlanie rekordów w bazie danych
    - wykorzystuje silnik Thymeleaf w warstwie prezentacji
    - wykorzystuje bazę danych H2
    - aplikacja napisana bez Spring Boot zawiera dużą ilość dodatkowego kodu - konfiguracja







# 2

## **Działanie Spring Boot**


# Działanie Spring Boot

- 
- **Przykład 2:** Minimalna aplikacja w Spring Boot
  - Dwa główne elementy:
    - adnotacja `@SpringBootApplication`
    - klasa `SpringApplication`
  - Klasa `SpringApplication` odpowiada m.in. za uruchomienie aplikacji i stworzenie instancji `ApplicationContext`.
  - W dalszej części skupimy się na adnotacji `@SpringBootApplication`.


# Adnotacja @SpringBootApplication

- 
- Umieszczona zwykle na poziomie głównej klasy w aplikacji  
Równoważna trzem innym adnotacjom:  
@Configuration  
@EnableAutoConfiguration  
@ComponentScan
  - Możemy to zobaczyć dzięki opcji Open Declaration w Eclipse.
  - Te adnotacje są częścią frameworka Spring
  - Aby nie wchodzić zbyttnio w szczegóły działania samego Springa skupimy się na drugiej i trzeciej.

# Adnotacja @ComponentScan

- 
- odpowiada za skanowanie w celu poszukiwania w projekcie komponentów (Spring Bean)
  - użycie jej bez atrybutów oznacza: znajdź komponenty w tym pakiecie oraz wszystkich pod-pakietach
  - komponenty znalezione w ten sposób mogą być następnie m.in. wstrzykiwane za pomocą adnotacji @Autowired
  - daje dużo możliwości, m.in. pozwala zdefiniować, które komponenty powinny zostać pominięte - my skupimy się na przypadku bazowym

# Adnotacja @EnableAutoConfiguration

- 
- Wprowadza do działania system automatycznej konfiguracji.
  - Celem jest dokonanie przez framework automatycznej konfiguracji aplikacji na podstawie zawartości *classpath*.
  - Mechanizm jest nieinwazyjny - automatyczną konfiguracja jest wdrażana tylko wtedy, gdy spełnione są odpowiednie warunki: dodaliśmy odpowiednie zależności i nie nadpisaliśmy konfiguracji sami.
  - Takie podejście sprawia, że chcąc zrobić coś niestandardowo nie musimy „walczyć” z frameworkiem.
  - **Przykład 3:** Adnotacje @ComponentScan i @EnableAutoConfiguration

# Spring Boot Starters

- Zbiór wygodnych deskryptorów zależności  
Przykład dla Maven:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```


- Każda wersja Spring Boot wspiera konkretne wersje zależności tak, by nie pojawiały się żadne konflikty.



3


## **Spring Boot Actuator**

# Spring Boot Actuator

- 
- Narzędzie pozwalające monitorować i zarządzać aplikacją m.in. za pomocą specjalnego zestawu endpointów HTTP.
  - Możemy monitorować np.: stan aplikacji (ang. *health*), listę komponentów wchodzących w skład aplikacji czy listę endpointów aplikacji.
  - W pliku `application.properties` możemy łatwo konfigurować dostępność tych endpointów - nie wszystkie dostępne są domyślnie (od wersji 2.0 większość nie jest).
  - Endpoint bazowy: `/actuator`
  - Rozbudowane narzędzie, tutaj zaprezentowane tylko podstawy.
  - **Przykład 4:** użycie Spring Boot Actuator na prostej aplikacji.



# Spring Boot Actuator

- 
- Niektóre endpointy:
    - **health** - podsumowanie stanu naszej aplikacji
    - **shutdown** - wyłączenie aplikacji
    - **info** - ogólne informacje
    - **beans** - lista komponentów Spring-owych
    - **logfile** - logi aplikacji
    - **metrics** - szczegółowe metryki aplikacji
    - **mappings** - mapowania ścieżek (@RequestMapping)
    - **httptrace** - ostatnie zapytania HTTP (konfiguracja)

Istnieje możliwość tworzenia własnych endpointów jak również rozszerzania funkcjonalności tych domyślnych.



# 4

## **Logowanie zdarzeń w aplikacji**



## SLF4J

SLF4J to jedna z najpopularniejszych bibliotek stosowanych w środowisku Java, która umożliwia logowanie zdarzeń występujących w trakcie działania aplikacji. Spring Boot zaadoptował to rozwiązanie jako domyślne narzędzie do sporządzania logów aplikacji.

# Logging levels

## TRACE

Najbardziej szczegółowy poziom logowania. Idealnie nadaje się do zapisywania wszystkich możliwych aktywności systemu.

## DEBUG

Szczegółowe logowanie informacji o zdarzeniach. Stosuję się najczęściej do debugowania kodu.

## INFO

Zapewnia informacje o akcjach zachodzących w systemie, tych które faktycznie zostały wykonane i kiedy zostały wykonane.

## WARN

Informuje o podejrzanych akcjach jakie nastąpiły podczas pracy aplikacji.

## ERROR

W logach ERROR ładują wszystkie przechwycone wyjątki, zarówno systemowe jak i zdefiniowane.

# Podstawowe ustawienia

## Pom.xml

Aby móc korzystać z logowania należy dodać odpowiednią zależność.

Jeśli korzystamy ze starteru starter-web to dostarczony zostanie również starter do logowania.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Jeśli nie korzystamy z dependencji starter-web to należy dołączyć bezpośrednio starter-logging.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-logging</artifactId>
</dependency>
```

# Wypisanie logów

Slf4j stanowi fasadę do logowania w aplikacji udostępniając wspólne API dla różnych bibliotek. Jednak aby wyświetlić logi należy podpiąć interesujący nas bibliotekę logującą (LogBack).

```
@Controller
@RequestMapping("/logger")
public class LoggerController {

    private static final Logger logger =
        LoggerFactory.getLogger(LoggerController.class);

    @RequestMapping(method = RequestMethod.GET)
    public String showLog(){

        logger.info("You can log something");
        logger.warn("You can log something");
        logger.debug("You can log something");
        logger.error("You can log something");

        return "some_page";
    }
}
```

# Logi a profile

- Spring Boot udostępnia możliwość konfiguracji logów w zależności od profilu.
- Np. w środowisku deweloperskim definiujemy logowanie wszystkich poziomów wraz z DEBUG na konsolę, a w środowisku produkcyjnym zapisujemy logi tylko do pliku.
- Aby mieć pełną kontrolę nad tym co i kiedy będzie logowane należy stworzyć plik logback-spring.xml w folderze src/main/resources projektu.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <include resource="org/springframework/boot/logging/logback/base.xml" />
  <springProfile name="prod">
    <logger name="pl.codecouple.omomfood.offers" level="ERROR" additivity="false">
      <appender-ref ref="FILE" />
    </logger>
  </springProfile>
  <springProfile name="dev">
    <logger name="pl.codecouple.omomfood.offers" level="DEBUG" additivity="false">
      <appender-ref ref="CONSOLE" />
    </logger>
  </springProfile>
</configuration>
```

# application.properties

- application.properties również umożliwia określenie podstawowych parametrów logowania bez konieczności tworzenia pliku logback-spring.xml.
- Wykorzystując plik application.properties możliwe jest zdefiniowanie m.in. następujących wartości:

## logging.level.\*

- oczekiwany poziom logowania
- logging.level.org.springframework.web=DEBUG

## logging.file || logging.path

- zdefiniowanie ścieżki do pliku
- jeśli ustawimy obie wartości, czyli oraz zostaną one zignorowane.

## logging.pattern.console || logging.pattern.file

- określenie wzorca jaki mają przyjąć logowane wiadomości

## logging.config

- ścieżka do pliku xml z konfiguracją

## spring.profiles.active

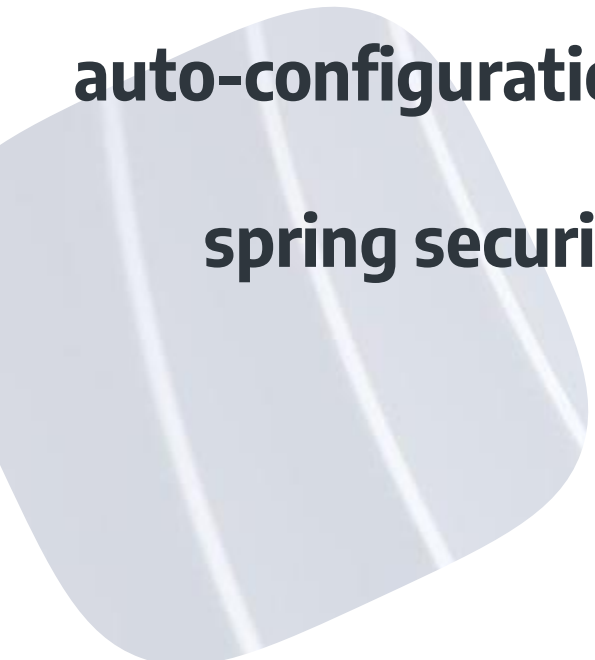
- określenie profilu





5

**Spring Security**



## **auto-configuration + spring security**

W większości przypadków predefiniowana konfiguracja środowiska Spring Boot jest wystarczająca. Jednak istnieją przypadki, w których należy ją nadpisać. Takim przypadkiem jest aplikacja, w której chcemy wykorzystać mechanizmy zapewniane przez Spring Security.



## security starter

- Pakiet Spring Boot udostępnia dependencję zapewniającą uzupełnienie aplikacji o mechanizmy zabezpieczeń.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

- Dodanie powyższej dependencji do pliku pom.xml spowoduje wyświetlenie monitu wymagającego podania danych autoryzujących użytkownika w przeglądarce.



## overwriting auto-configuration

- Nadpisanie automatycznej konfiguracji wymaga zaimplementowania jej od nowa, jak gdyby nigdy wcześniej nie istniała. Aby tego dokonać należy nadpisać klasę rozszerzającą funkcjonalności `WebSecurityConfigurerAdapter`.

```
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter
```



## overwriting auto-configuration

- Klasy rozszerzające `WebSecurityConfigurerAdapter` mogą nadpisywać dwie różne metody `configure()`.
- Pierwsza z nich określa, dla jakiej ścieżki wymagane jest uwierzytelnienie użytkownika. Zasoby umieszczone pod wszystkimi innymi ścieżkami będą otwarte dla wszystkich.
- Druga definiuje jaki standard połączenia z bazą danych będzie stosowany w aplikacji.



6

**Spring Boot Test**



# Spring Boot Test

- Spring Boot Test to zestaw narzędzi ułatwiających testowanie aplikacji. Dodając do zależności aplikacji moduł starter-test otrzymujemy cały zestaw użytecznych bibliotek, wśród których możemy znaleźć:
  - **Junit** – podstawowa biblioteka do uruchamiania testów,
  - **Mockito** – biblioteka do tworzenia mockowych obiektów,
  - **Spring Test i Spring Boot Tests** – zestaw narzędzi do testowania Spring i Spring Boota.



## Adnotacje

- **@RunWith**

- Informuje silnik testów (np. JUnit), aby uruchomił dany test z użyciem SpringRunniera. Dzięki temu framework wie, że to będzie test z wykorzystaniem Springa i powinien wstrzyknąć zależności.

- **@WebMvcTest**

- Informuje, że chcemy testować komponenty MVC, więc skonfigurowane zostaną dodatkowe elementy, takie jak mockMvc. Zostanie skonfigurowany cały kontekst springowy, ale nie będzie tworzony serwer – test uruchomi się szybciej.





## Adnotacje

- **@SpringBootTest**
  - Informuje silnik testów, że testowany będzie cały kontekst springa, nie tylko MVC.
- **@AutoConfigureMockMvc**
  - Zachowanie możliwości korzystanie z MockMvc
- **@MockBean**
  - tworzenie obiektu mock, zamiast prawdziwego serwisu



## Sposoby testowania

1. Wstrzyknięcie kontrolera do testu i **wywołanie metody**.
2. Wykorzystanie **MockMVC** i wykonanie zapytania. Pozwala dodatkowo na sprawdzenie, czy adres zostaje prawidłowo mapowany oraz, czy serializacja danych działa zgodnie z oczekiwaniami.
3. Mockowanie serwisu za pomocą adnotacji **@MockBean**.



## Źródła

- Craig Walls - *Spring Boot in Action*
- Craig Walls - *Spring in Action*
- Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho - *Pro Spring 5: An In-Depth Guide to the Spring Framework and Its Tools, Fifth Edition*
- Jakub Kubryński - prezentacja Spring Boot ([link](#))
- Oficjalna dokumentacja ([link](#))
- Strona mkyong.com ([link](#))
- Strona baeldung.com ([link](#))