



1

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 łądzą 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

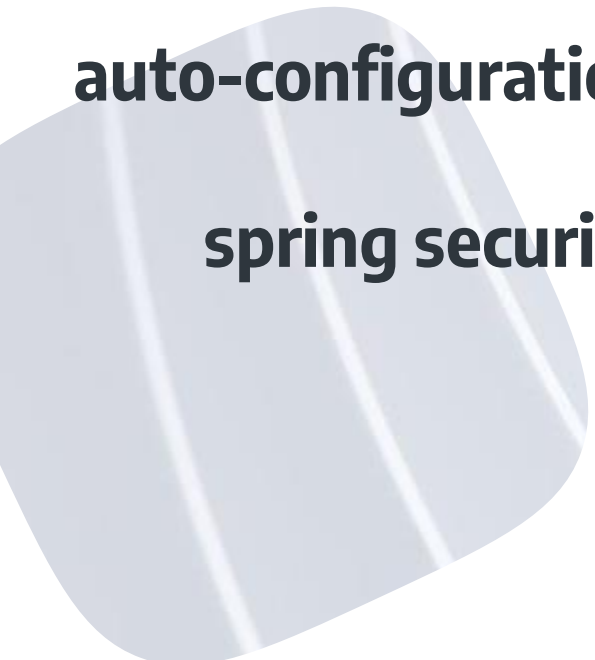
spring.profiles.active

- określenie profilu



2

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 dependencje 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 on 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 wymagana jest uwierzytelnienie użytkownika. Zasoby umieszczone pod wszystkimi innymi ścieżkami będzie otwarte dla wszystkich.
- Druga definiuje jaki standard połączenia z bazą danych będzie stosowany w aplikacji.



3

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**.