SpringBoot camp

:docInfo1

Table of Contents

1. O mnie 1
2. Moje źródła wiedzy 1
3. Co to jest Spring Boot ? 1
4. Moduły 1
5. Cechy frameworka 3
6. Dokumentacja 7
7. Narzędzia i generowanie rusztowania Spring Boot 7
    7.1. Spring Boot CLI 7
    7.2. STS 7
    7.3. IDEA 7
    7.4. Initializr 7
8. Spring Boot 1.4 features 7
    8.1. Startup Error Improvements 7
    8.2. Actuator Improvements 7
    8.3. wsparcie dla testów 7
    8.4. Datasource binding / wybieramy dostarczyciela ds 9
    8.5. Image Banners 9
    8.6. Third Party Libraries 9
    8.7. Spring 4.3.0 Support 10
9. Strona projektu 11
10. Spring vs Spring Boot 11
    10.1. Podejście standardowe 11
    10.2. Aplikacja WEB/REST 11
11. Maven 12
    11.1. Konfiguracja 12
    11.2. Tworzenie wykonywalnego jar’a 12
    11.3. Dziedziczenie z 'parenta' 13
    11.4. Spring Boot bez udziału parent POM 13
    11.5. Zmiana wersji javy 14
    11.6. Użycie pluginu Spring Boot 14
    11.7. Wyłącznie biblioteki 14
    11.8. Włączenie nowszej wersji biblioteki 15
    11.9. Uruchamianie 15
12. Gradle 15
    12.1. Konfiguracja 15
    12.2. spring-boot-gradle-plugin 16
    12.3. Wyłączanie biblioteki 17
    12.4. Włączanie nowszej wersji 17

12.5. Uruchamianie . . . 17
13. Java . . . 17
14. Uruchamianie jako jar package . . . 18
15. Starters . . . 18
16. @Conditional . . . 21
17. Konfiguracja . . . 24
17.1. Blokowanie wybranej autokonfiguracji . . . 25
17.2. @SpringBootApplication . . . 25
17.3. Lokalizowanie kandydatów do auto-konfiguracji . . . 25
18. @Enable . . . 25
19. Konfigurowanie . . . 25
19.1. Lokalizowanie kandydatów do auto-konfiguracji . . . 25
19.2. Porządek konfiguracji . . . 26
19.3. Class conditions . . . 26
19.4. Bean conditions . . . 26
19.5. Property conditions . . . 26
19.6. Web application conditions . . . 26
19.7. SpEL expression conditions . . . 26
19.8. Tworzenie własnego startera . . . 26
19.9. Przykład . . . 27
19.10. Health . . . 27
19.11. XML a Spring Boot . . . 28
20. Debug mode . . . 28
21. Hot swapping . . . 28
22. Banner . . . 28
22.1. Fluent builder API . . . 29
23. Application events and listeners . . . 29
24. Argumenty wywołania . . . 30
25. Parametry z 'command line' . . . 30
26. Admin features . . . 30
27. Konfigurowanie zewnętrzne . . . 30
27.1. Zmiana lokacji pliku ustawień i jego nazwy . . . 31
27.2. Random . . . 31
27.3. Placeholders w pliku properties . . . 32
27.4. Automatyczne wstawianie danych przez proces budujący → Maven . . . 32
27.5. Automatyczne wstawianie danych przez proces budujący → Gradle . . . 33
27.6. YAML . . . 33
27.7. ConfigurationProperties . . . 34
27.8. Type-safe Configuration Properties . . . 34
27.9. Relaxed binding . . . 35
27.10. @ConfigurationProperties walidacje . . . 36

27.11. Meta-dane / Wsparcje dla IDE . . . 36
27.12. Value . . . 36
27.13. PropertySource . . . 38
28. Profile . . . 39
28.1. Programowalne ustawianie profilu . . . 39
29. CommandLineRunner . . . 39
30. ApplicationArguments . . . 40
31. DevTools . . . 40
31.1. Tuning . . . 41
31.2. LiveReload plugin . . . 41
31.3. Uniemożliwienie restartów . . . 41
31.4. Zdalne aplikacje . . . 41
32. Spring loaded . . . 42
33. Logowanie . . . 42
33.1. Console . . . 42
33.2. Pliki z logami . . . 43
33.3. Poziomy logowania . . . 43
33.4. Konfiguracja logów użytkownika . . . 43
34. WEB . . . 43
34.1. Aktualne kontenery . . . 43
34.2. Spring MVC auto-configuration . . . 43
34.3. Serwer port . . . 44
34.4. Static Content . . . 44
34.5. Template engines . . . 44
34.6. favicon . . . 44
35. Defiowanie filtrów . . . 45
36. Interceptory . . . 45
37. PropertyEditors . . . 45
38. Formatter . . . 46
39. Defiowanie ścieżek do własnych zasobów statycznych . . . 47
40. Jetty zamiast Tomcat’a . . . 48
41. Error Handling . . . 48
42. SimpleMappingExceptionResolver . . . 53
42.1. Definiowanie stron błędów . . . 54
43. Rest . . . 55
43.1. Adnotacje . . . 55
43.2. Prefix usługi . . . 55
43.3. Spring Data REST . . . 55
43.4. Wywoływanie serwisu REST . . . 56
43.5. Dostrojenie RESTTemplate . . . 56
43.6. Richardson Maturity Model . . . 57

43.7. Hal browser . . . 57
43.8. Swagger 2 . . . 58
43.9. Obsługa wyjątków . . . 59
43.10. Hateoas . . . 60
44. JSON . . . 64
44.1. @JsonView . . . 64
44.2. @JsonBackReference . . . 65
44.3. @JsonManagedReference . . . 66
44.4. @JsonPropertyOrder . . . 66
44.5. @JsonSerialize . . . 66
44.6. @JsonCreator - serializacja . . . 67
45. SSL . . . 68
45.1. Generacja klucza . . . 68
45.2. Ustawienia . . . 68
46. Context path . . . 71
47. Dalsza optymalizacja . . . 71
47.1. Zasoby statyczne : . . . 71
47.2. Cache-Control, ETag, Last-Modified . . . 72
47.3. Parametryzowanie: . . . 72
47.4. Gzip . . . 73
47.5. Cache + @Cacheable . . . 73
47.6. Cache Guava . . . 73
48. ORM . . . 74
48.1. Ładowanie danych z skryptu SQL . . . 75
48.2. Ładowanie danych programowe poprzez repozytoria . . . 75
48.3. www.liquibase.org[Liquibase] . . . 75
48.4. Flyway . . . 76
48.5. JNDI . . . 76
48.6. JdbcTemplate . . . 76
48.7. JPA & Spring Data . . . 77
48.8. Użycie H2’s web console . . . 77
48.9. Zabezpieczenie h2-console . . . 77
49. Źródła wiedzy . . . 78
50. Spring Data / Hibernate / JPA . . . 78
50.1. Najnowsze features:) . . . 78
50.2. Geneza . . . 78
50.3. Charakretystyka . . . 79
50.4. Narzędzia . . . 79
50.5. Nazwane zapytania . . . 84
50.6. Web support . . . 94
51. NoSql . . . 97

52. MongoDB 97
52.1. Przykładowa encja 97
52.2. Parametryzacja 98
52.3. MongoTemplate 98
52.4. Konfiguracja 98
52.5. Spring Data MongoDB repozytoria 99
52.6. Embedded Mongo 100
53. Neo4j 101
53.1. Połączenie z bazą Neo4j 101
53.2. Neo4jTemplate 101
53.3. Embedded mode 102
53.4. Spring Data Neo4j repozytoria 102
54. Using ResourceBundles for Internationalization (I18N) 102
55. Spring Security 102
55.1. OWASP Top Ten 103
55.2. Authentication : autoryzacja: uwierzytelnianie 103
55.3. Access control: Kontrola dostępu 105
55.4. Standartowe podejście 105
55.5. Maven: 105
55.6. Gradle : 106
55.7. Moduły 106
55.8. Sposoby logowania 106
55.9. Zabezpieczanie dostępu do adresów URL 107
55.10. Zabezpiecznie ścieżek dostępu 107
55.11. Zabezpieczanie za pomocą SpEL : zwiększamy elastyczność 108
56. Koncepcja 108
56.1. UserDetailService (read-only) 110
56.2. UserDetails 110
56.3. @EnableWebSecurity 111
56.4. Login Form 111
56.5. Zabezpieczanie widoków widoków : Thymeleaf 112
56.6. Backend 113
56.7. @EnableGlobalMethodSecurity 114
57. Caching 114
57.1. Możliwi dostawcy 114
57.2. CacheManager 114
57.3. EhCache 2.x 114
58. Caffeine / Guava 116
59. Messaging 116
60. JMS 116
60.1. Rodzaje : 116

60.2. Typy konsumentów : . . . 117
60.3. JmsTemplate . . . 117
61. ActiveMQ support . . . 117
61.1. Sterowanie komunikatami **@MessageDriven** . . . 124
61.2. @EnableJms . . . 125
61.3. Wysyłanie wiadomości . . . 125
61.4. Odbieranie wiadomości . . . 125
61.5. Własny JmsListenerContainer . . . 126
61.6. Monitoring . . . 126
62. AMQP . . . 127
62.1. Wymiana komunikatów . . . 128
62.2. Komponenty . . . 129
62.3. Exchange Type . . . 129
62.4. RabbitMQ support . . . 132
62.5. Wysyłanie wiadomości . . . 139
62.6. Odbieranie wiadomości . . . 140
62.7. Własny RabbitListenerContainerFactory . . . 140
63. Przykłady . . . 141
63.1. Równoległy odbiorca . . . 141
63.2. Pub/Sub . . . 143
63.3. Routing . . . 145
64. Actuator . . . 147
64.1. Zależność . . . 147
64.2. Kanały . . . 147
64.3. Co możemy zrobić z actuator endpoints ? . . . 148
64.4. HealthEndpoint . . . 149
64.5. Deaktywacja wszystkich domyślnych . . . 150
64.6. Sortowanie . . . 150
64.7. Własny Health endpoint / HealthIndicator . . . 150
64.8. InfoContributors . . . 151
64.9. Zmienne aktywne w czasie budowanie projektu . . . 152
64.10. remote shell . . . 155
64.11. Własne metryki . . . 156
64.12. Trwałe metryki . . . 157
64.13. Dostrajanie portów . . . 160
64.14. Zmiana nazwy endpointu . . . 160
64.15. Rozszerzenia : . . . 160
64.16. Zamykanie aplikacji . . . 160
64.17. Zabezpiecznie / Security . . . 160
64.18. Security / denial of service . . . 161
64.19. Spring Boot Admin . . . 161

64.20. Dropwizard 162
65. Sending email 163
65.1. Zależność 163
65.2. Parametryzacja 164
66. Testing 164
66.1. Moduł 164
66.2. Zależności dodane : 164
66.3. Testy integracyjne 164
67. Spring 1.4 usprawnienia 165
67.1. @RunWith(SpringRunner.class) 165
67.2. Mock 166
67.3. @JsonTest 167
67.4. Kontrolery 167
67.5. Dodanie filtrów 168
67.6. JPA 169
67.7. Testy klientów REST 169

1. O mnie

- Architect Solution - RiscoSoftware
- VavaTech trener : Spring ekosystem, JPA , EIP Camel
- Sages trener : JPA , EIP - Apache Camel
- blog http://przewidywalna-java.blogspot.com
- twitter przodownikR1

[tUWf7KiC]

2. Moje źródła wiedzy

- spring boot project
- spring boot reference
- **Moje Książki**
  - Learing Spring Boot
  - Spring Boot in Action
  - Pro Spring Boot
  - Spring Boot CookBook
  - Building MicroServices
  - The_Tao_of_MicroServices (MEP)
  - Spring MicroServices
  - Testing Java MicroServices (MEP)
  - Mastering MicroServices with JAVA
  - Spring MVC Cookbook
  - Spring Security 3.1
  - Mastering Spring MVC 4
  - Building A Restful Web Service with Spring
  - Spring REST
  - Mockito for Spring

3. Co to jest Spring Boot ?

4. Moduły

- **Spring Boot**

- **Spring Boot Autoconfigure**

  ```
  @EnableAutoConfiguration, tworzenie kontekstu
  ```

- **Spring Boot Starters**

  ```
  zależności
  ```

- **Spring Boot CLI**
  - groovy
  - java
  - java command line runner

Przykładowy kontroller - groovy

```
@RestController
class Simple{

@RequestMapping("/hello")
 String hello(){
 return "hello boot"
 }
}
```

Przykładowe wywołanie :

```
spring run simpleController.groovy -- --server.port=9009
```

- **Spring Boot Actuator**

  ```
  production ready features
  ```

  - auto-config
  - konwencje
  - wbudowany kontener servletów
  - metryki
  - logi
  - funkcje 'życiowe'

- **Spring Boot Tools**

```
budowanie jar / war
```

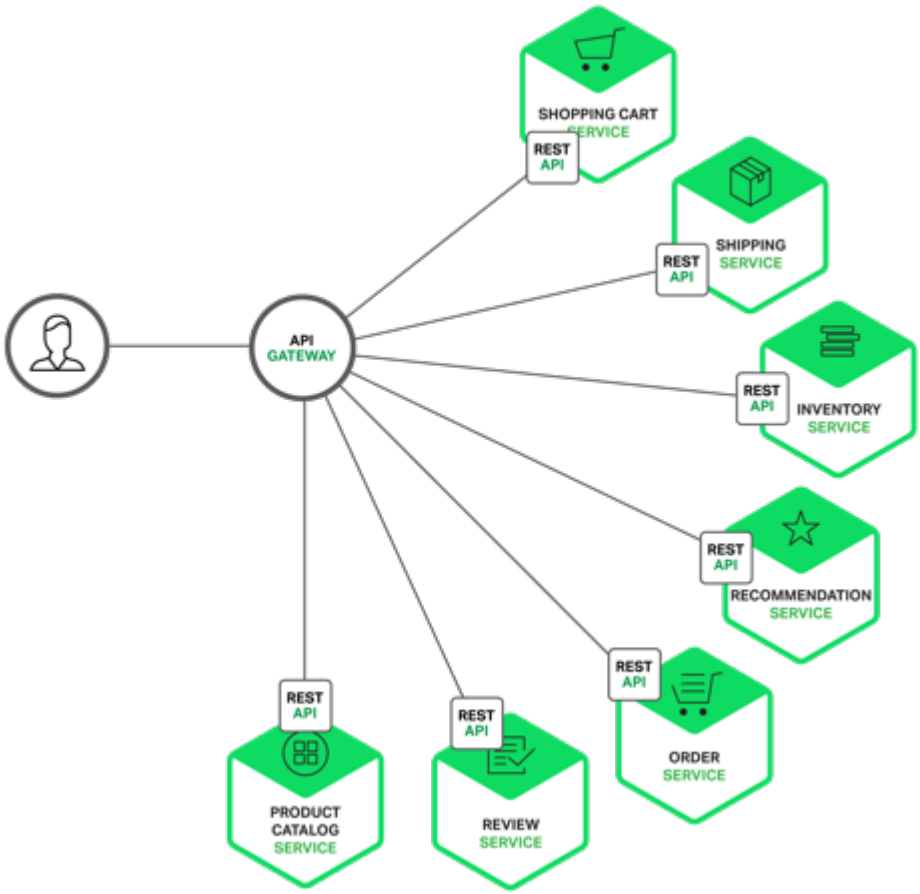
- **Spring Boot Samples**

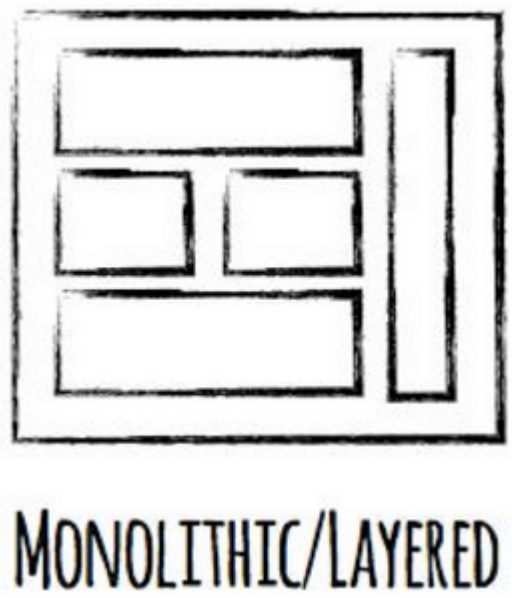
```
przykłady
```

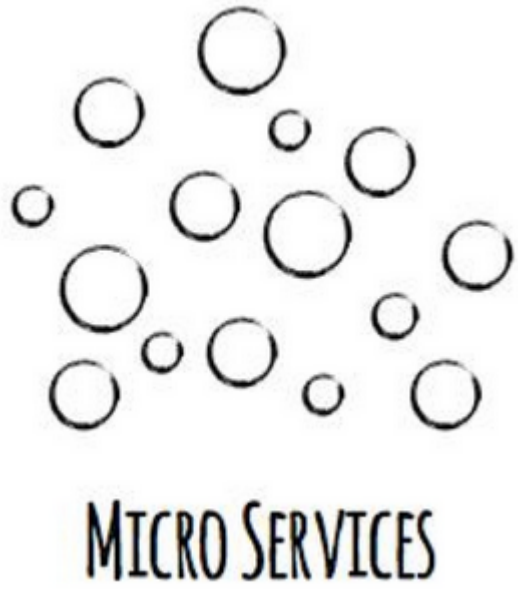
5. Cechy frameworka

- Błyskawiczne wytwarzanie
- Prototypowanie
- Eliminacja XML
- Autokonfiguracja
  - Prostota konfiguracji zależności
  - Prostota konfiguracji aplikacji
- Starters dependencies
- Command-line interface
- Wbudowany kontener Tomcat lub Jetty
- Zcentralizowana konfiguracja
  - application.properties
  - application.yml
- Profile
- JAR !
- Monitoring
  - Actuator
- Microservices :)

source: https://www.nginx.com/blog/building-microservices-inter-process-communication/


SHOPPING CART
SERVICE
REST
API
SHIPPING
SERVICE
REST
API
API
GATEWAY
REST
API
INVENTORY
SERVICE
REST
API
RECOMMENDATION
SERVICE
REST
API
ORDER
SERVICE
REST
API
PRODUCT
CATALOG
SERVICE
REST
API
REVIEW
SERVICE





Za :

- łatwość wytwarzania aplikacji typu 'stand-alone' gotowych na wdrożenie produkcyjne
- szybka konfiguracja wykorzystująca model Convention over Configuration
- możliwość wyrzucenia konfiguracji poza projekt
- wczechobecny jar zamiast starego ear czy war
- szybki deploy
- wielkie serwery aplikacyjne
- monolit - utrzymywanie i rozwijanie
- splątane zależności
- małe autonomiczne usługi
- małe autonomiczne zespoły
- łatwość zrozumienia domeny
- łatwa skalowalność
- izolacja sytuacji krytycznych
- dobór odpowiednich narzędzi i języków do danej domeny

Przeciw :

- testowanie
- debugowanie
- próg wejścia
- nowe wzorce architektoniczne

- zwiększona pamięciożerność

Aktualna wersja 1.4.0.RELEASE wymaga Java 7 i Spring Framework 4.3.2.RELEASE lub wyżej

6. Dokumentacja

dokumentacja

7. Narzędzia i generowanie rusztowania Spring Boot

7.1. Spring Boot CLI

```
spring init -dweb,data-jpa,h2,thymeleaf --build gradle mySimpleApp --force
```

7.2. STS

7.3. IDEA

7.4. Initializr

8. Spring Boot 1.4 features

8.1. Startup Error Improvements

```
krótszy i przejrzysty **stacktrace** w przypadku błędu
```

8.2. Actuator Improvements

- informacje o Git generowane prostu z pluginów maven czy gradle
- nowe konfiguracje dla : Neo4j, Caffeine Cache (guava), CouchBase, etc

8.3. wsparcie dla testów

- **@SpringBootTest**
- **@WebMvcTest** - testowanie warstwy WEB/Controller

```
    @RunWith(SpringRunner.class)
    @WebMvcTest(MyController.class)
    public class MyControllerTests {

        @Autowired
        private MockMvc mvc;

        @MockBean
        private MyService myService;

        @Test
        public void getSource() throws Exception {
            given(myService.getLastSource()).willReturn(new Source("ok let's do this"
));

            mvc.perform(get("/").accept(MediaType.APPLICATION_JSON))
                    .andExpect(status().isOk())
                    .andExpect(content().json("{'source':'ok let's do this'}"));
        }
    }
```

- **@JsonTest** - testowanie marshalling/unmarshalling dla Jsona

```
```

- **@DataJpaTest** - testowanie JPA

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class ExampleRepositoryTests {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private UserRepository repository;

    @Test
    public void testExample() throws Exception {
        this.entityManager.persist(new User("sboot", "1234"));
        User user = this.repository.findByUsername("sboot");
        assertThat(user.getUsername()).isEqualTo("sboot");
        assertThat(user.getVin()).isEqualTo("1234");
    }

}
```

- **@RestClientTest**

```
@RunWith(SpringRunner.class)
@RestClientTest(RemoteVehicleDetailsService.class)
public class ExampleRestClientTest {

    @Autowired
    private RemoteVehicleDetailsService service;

    @Autowired
    private MockRestServiceServer server;

    @Test
    public void getVehicleDetailsWhenResultIsSuccessShouldReturnDetails()
            throws Exception {
        this.server.expect(requestTo("/greet/details"))
                .andRespond(withSuccess("hello", MediaType.TEXT_PLAIN));
        String greeting = this.service.callRestService();
        assertThat(greeting).isEqualTo("hello");
    }

}
```

8.4. Datasource binding / wybieramy dostarczyciela ds

```
spring.datasource.hikari.maximum-pool-size=5
spring.datasource.hikari.connection-timeout=10
```

8.5. Image Banners

```
banner jako rysunek
```

8.6. Third Party Libraries

```
wsparcie nowych bibiliotek
```

- Hibernate 5.1
- Jackson 2.7
- Solr 5.5
- Spring Data Hopper
- Spring Session 1.2
- Hazelcast 3.6.

8.7. Spring 4.3.0 Support

```
oparty na Spring'u 4.3.0
```

- niejawny @Autowired na konstuktorze
- możliwość definiowania default method
- nowe meta-dane dla cache oraz wsparcie dla długo działających procesów

```
@Cacheable(cacheNames = "foos", sync = true)
public Foo getFoo(String id) { ... }
```

- uproszczone API dla opercji REST/WEB
  - @GetMapping = @RequestMapping(method = RequestMethod.GET)
  - @PostMapping
  - @PutMapping
  - @DeleteMapping
  - @PatchMapping
- @RequestScope, @SessionScope, @ApplicationScope // określenie nowych zakresów
- @RequestAttribute, @SessionAttribute // dostep do atrybutów sesji i requestu

```
@GetMapping
public String getPerson(@SessionAttribute String login, @RequestAttribute String
query) {
   ..//
}
```

- wsparcje dla nowych wersji bibliotek
  - Hibernate ORM 5.2
  - Jackson 2.8 min Jackson 2.6+ dla Spring 4.3
  - Netty 4.1
  - Undertow 1.4
  - Tomcat 8.5.2 oraz wczesne wesje 9
  - wsparcie dla Hibernate 5

9. Strona projektu

10. Spring vs Spring Boot

10.1. Podejście standardowe

- Projekt webowy musi zawierać folder WEB-INF
  - opjonalny plik : web.xml
  - <servlet-name>-servlet.xml

10.2. Aplikacja WEB/REST

- Stworzenie pliku web.xml lub WebApplicationInitializer, żeby zadeklarować DispatcherServlet
- Skonfigurowanie Spring MVC
- Stworzenie prostego kontrolera i zarejestrowanie go
- Odpalenie aplikacji na jakimś kontenerze servletów lub serwerze aplikacyjnym
  - Przykład / Prosty kontroller

```
@Controller
@EnableAutoConfiguration
public class SampleController {

    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!";
    }

    public static void main(String[] args) throws Exception { //configuration &
bootstrapping
        SpringApplication.run(SampleController.class, args);
    }
}
```

```
@RestController
public class SimpleWebController {
@RequestMapping("/")
public String greetings(){
return "Hello world";
}
}
```

11. Maven

11.1. Konfiguracja

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>
<groupId>com.example</groupId>
<artifactId>myapp</artifactId>
<version>0.0.1-SNAPSHOT</version>

<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.6.RELEASE</version>
</parent>

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

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

<build>
<plugins>
  <plugin>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-maven-plugin</artifactId>
   </plugin>
</plugins>
</build>
</project>
```

11.2. Tworzenie wykonywalnego jar’a

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

11.3. Dziedziczenie z 'parenta'

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.0.RELEASE</version>
</parent>
```

```
<properties>
    <spring-data-releasetrain.version>Fowler-SR2</spring-data-releasetrain.version>
</properties>
```

11.4. Spring Boot bez udziału parent POM

```
<dependencyManagement>
     <dependencies>
        <dependency>
            <!-- Import dependency management from Spring Boot -->
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>1.4.0.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.data</groupId>
            <artifactId>spring-data-releasetrain</artifactId>
            <version>Fowler-SR2</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-dependencies</artifactId>
            <version>1.4.0.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

11.5. Zmiana wersji javy

```
<properties>
    <java.version>1.8</java.version>
</properties>
```

11.6. Użycie pluginu Spring Boot

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

11.7. Wyłącznie biblioteki

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
            <exclusion>
                <groupId>com.fasterxml.jackson.core</groupId>
            </exclusion>
    </exclusions>
</dependency>
```

11.8. Włączenie nowszej wersji biblioteki

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.4.8</version>
</dependency>
```

11.9. Uruchamianie

```
mvn spring-boot:run

$ export MAVEN_OPTS=-Xmx1024m -XX:MaxPermSize=128M
```

12. Gradle

12.1. Konfiguracja

```
buildscript {
repositories {
jcenter()
maven { url "http://repo.spring.io/snapshot" }
maven { url "http://repo.spring.io/milestone" }
}
dependencies {
classpath("org.springframework.boot:spring-boot-gradle-plugin:1.3.6.RELEASE")
}
}

apply plugin: 'java'
apply plugin: 'spring-boot'
jar {
   baseName = 'mySimpleApp'
   version = '0.0.1-SNAPSHOT'
}
repositories {
   jcenter()
   ...
}
dependencies {
   compile 'org.springframework.boot:spring-boot-starter'
   compile("org.springframework.boot:spring-boot-starter-web")
   testCompile("org.springframework.boot:spring-boot-starter-test")
}
```

```
apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web:1.4.0.RELEASE")
}
```

12.2. spring-boot-gradle-plugin

```
buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:1.4.0.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'spring-boot'

repositories {
    jcenter()
}

dependencies {
    compile("org.springframework.boot:spring-boot-starter-web")
    testCompile("org.springframework.boot:spring-boot-starter-test")
}
```

12.3. Wyłączanie biblioteki

```
compile("org.springframework.boot:spring-boot-starter-web") {
exclude group: 'com.fasterxml.jackson.core'
}
```

12.4. Włączanie nowszej wersji

```
compile("com.fasterxml.jackson.core:jackson-databind:2.4.8")
```

12.5. Uruchamianie

```
gradle bootRun

export JAVA_OPTS=-Xmx1024m -XX:MaxPermSize=128M
```

13. Java

```
spring run *.java
```

14. Uruchamianie jako jar package

$ java -jar target/myproject-0.0.1-SNAPSHOT.jar

$ java -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=8000,suspend=n -jar target/myproject-0.0.1-SNAPSHOT.jar

15. Starters

- **spring-boot-starter**

Rdzeń Spring Boot. Zawiera elementy decydujące o auto-konfiguracji, logowaniu i rejestrowaniu zmian w YAML czy properties.

Zależność krytyczna

- **spring-boot-starter-actuator**

Pomaga monitorować i zarządzać aplikacją.

Production ready feature

- **spring-boot-starter-amqp**

Wspiera technologię opartą protokole AMPQ jak RABBIT-MQ

- **spring-boot-starter-aop**

Wspiera AOP

- **spring-boot-starter-batch**

Wspiera integrację ze Spring Batch

- **spring-boot-starter-cache**

Wspiera integrację z Cache

- **spring-boot-starter-cloud-connectors**

Wspiera integrację z chmurą

- **spring-boot-starter-data-elasticsearch**

Wspiera integrację z ElasticSearch

- **spring-boot-starter-data-gemfire**

Wspiera integrację z GemFire

- **spring-boot-starter-data-jpa**

Wspiera integrację z JPA oraz Spring-Data-JPA (spring-data-jpa,spring-orm)

- **spring-boot-starter-data-mongodb**

Wspiera integrację z MongoDB (spring-data-mongodb)

- **spring-boot-starter-data-rest**

Wspiera integrację z REST (spring-data-rest-webmvc)

- **spring-boot-starter-data-solr**

Wspiera integrację z Solr (spring-data-solr)

- **spring-boot-starter-freemarker**

Wspiera integrację z FreeMarker

- **spring-boot-starter-groovy-templates**

Wspiera integrację z Groovy

- **spring-boot-starter-hateoas**

Wspiera integrację z HATEOAS (spring-hateoas)

- **spring-boot-starter-hornetq**

Wspiera integrację z HornetQ

- **spring-boot-starter-integration**

Wspiera integrację z Spring-integration

- **spring-boot-starter-jdbc**

Wspiera integrację z JDBC

- **spring-boot-starter-jersey**

Wspiera integrację z Jersey RESTful

- **spring-boot-starter-jta-atomikos**

Wspiera integrację JTA Atomikos

- **spring-boot-starter-jta-bitronix**

Wspiera integrację z JTA Bitronix

- **spring-boot-starter-mail**

Wspiera dla wysyłki email

- **spring-boot-starter-mobile**

Wspiera integrację z spring-mobile

- **spring-boot-starter-mustache**

Wspiera integrację z Mustache template

- **spring-boot-starter-redis**

Wspiera integrację z REDIS

- **spring-boot-starter-security**

Wspiera integrację z Spring-Security

- **spring-boot-starter-social-facebook**

Wspiera integrację z Facebook (spring-social-facebook)

- **spring-boot-starter-social-linkedin**

Wspiera integrację z LinkedIn (spring-social-linkedin)

- **spring-boot-starter-social-twitter**

Wspiera integrację z Twitter (spring-social-twitter)

- **spring-boot-starter-test**

Wspiera testowanie (Junit, Hamcrest, Mockito)

- **spring-boot-starter-thymeleaf**

Wspiera integrację z Thymeleaf template

- **spring-boot-starter-velocity**

Wspiera integrację z Velocity

- **spring-boot-starter-web**

Wsparcie dla stosu webowego Springa (Tomcat, spring-webmvc)

Wbudowany Tomcat, Jackson JSON binding, JSR 303 validation, Spring Web i Spring MVC

- **spring-boot-starter-websocket**

Wspiera integrację z WebSocket

- **spring-boot-starter-ws**

Wspiera integrację z Web Service

- **spring-boot-starter-actuator**

Monitoring i metryki

- **spring-boot-starter-remote-shell**

Monitoring i metryki z poziomu ssh

- **spring-boot-starter-jetty**

Integracja z Jetty (alternatywa dla Tomcat)

- **spring-boot-starter-log4j**

Wspiera integrację z Log4j

- **spring-boot-starter-logging**

Import Spring Boot's default logging framework (Logback).

- **spring-boot-starter-tomcat**

Wspiera integrację z Tomcat (domyślne)

- **spring-boot-starter-undertow**

Wspiera integrację z Undertow (alternatywa dla Tomcat)

16. @Conditional

- **Krok 1**

```
@Bean
  public OSService osService(){
    if (System.getProperty("os.name").contains("Windows")){
      return new WindowService();
    }
    return new LinuxService();
  }
}
```

- **Krok 2**

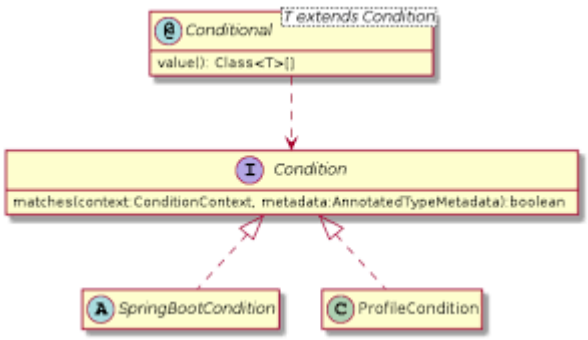
```
@Configuration
@Profile("Linux")
public class OsLinuxConfiguration {

  @Bean
  public LinuxService linuxService() {
    return new LinuxService();
  }
}


@Profile("Windows")
public class OsWindowsConfiguration {

  @Bean
  public WindowsService windowsService() {
    return new WindowsService();
  }
}
```

plus włączenie profilu

```
context.getEnvironment().setActiveProfiles("Linux");
```

- **Krok 3 → Conditional w praktyce**

source:blog.frankel.ch



Przykład

```
@Bean
@Target({ElementType.TYPE, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Conditional(AvailableOnClasspathCondition.class)
public @interface AvailableOnClasspath {
    public String value();
}
```

```
@Slf4j
class AvailableOnClasspathCondition implements Condition {

    @Override
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        Map<String, Object> attributes = metadata.getAnnotationAttributes
(AvailableOnClasspath.class.getName());
        String className = String.valueOf(attributes.get("value"));
        try {
            Class.forName(className);
            log.info("class exists on classpath");

            return true;
        } catch (ClassNotFoundException ignored) {

        }
        log.info("class not exists on classpath");
        return false;
    }

}
```

Przykład 2:

```
@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Conditional(ResourcePresentCondition.class)
public @interface ResourcePresentConditional {

    public String value();

    public boolean negated() default false;

}
```

```
@Slf4j
public class ResourcePresentCondition implements Condition {

    private static ResourcePatternResolver resourcePatternResolver = new
PathMatchingResourcePatternResolver();

    @Override
    public boolean matches(ConditionContext ctx, AnnotatedTypeMetadata metadata) {
        Map<String, Object> attributes = metadata
                .getAnnotationAttributes(ResourcePresentConditional.class
                        .getName());
        return matches((String) attributes.get("value"),
                (Boolean) attributes.get("negated"));
    }

    public static boolean matches(String value, boolean negated) {
        boolean b = false;
        try {
            Resource[] resources = resourcePatternResolver.getResources(value);
            for (Resource r : resources) {
                if (r.exists()) {
                    log.info("resource exists on the classpath");
                    b = true;
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return b && !negated || !b && negated;
    }

}
```

17. Konfiguracja

- Spring java config
  - Bean
  - @Configuration
  - @ImportResource
  - @ComponentScan
  - @Import
- Spring annotation
  - @ImportResource

    - @ComponentScan
    - @Import
- Spring profiles
- yaml/properties
- @SpringBootApplication - umożliwia skanowanie komponentów oraz auto-konfigurację

17.1. Blokowanie wybranej autokonfiguracji

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

- Przykład

```
@EnableAutoConfiguration(exclude=[ActiveMQAutoConfiguration.class])
```

To samo co :

```
@SpringBootApplication( exclude={ActiveMQAutoConfiguration.class} )
```

17.2. @SpringBootApplication

To samo co : @Configuration + @EnableAutoConfiguration + @ComponentScan

17.3. Lokalizowanie kandydatów do auto-konfiguracji

Spring Boot sprawdza plik **META-INF/spring.factories** w opublikowanych jarach. Taki plik powinien zawierać listę klas konfiguracyjnych oznaczonych @EnableAutoConfiguration

18. @Enable

19. Konfigurowanie

Rdzeń to klasa @Configuration + @Conditional oraz jej odmiany

META-INF/spring.factories

19.1. Lokalizowanie kandydatów do auto-konfiguracji

```
META-INF/spring.factories // w danym jar.
//lista configuracja powinna znajdować się poniżej klucza EnableAutoConfiguration
```

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.mycorp.libx.autoconfigure.LibXAutoConfiguration,\
com.mycorp.libx.autoconfigure.LibXWebAutoConfiguration
```

19.2. Porządek konfiguracji

- **@AutoConfigureAfter**
- **@AutoConfigureBefore**

19.3. Class conditions

- **@ConditionalOnClass**
- **@ConditionalOnMissingClass**

19.4. Bean conditions

- **@ConditionalOnBean**
- **@ConditionalOnMissingBean**

19.5. Property conditions

- **@ConditionalOnProperty**

19.6. Web application conditions

- **@ConditionalOnWebApplication**
- **@ConditionalOnNotWebApplication**

19.7. SpEL expression conditions

- **@ConditionalOnExpression**

19.8. Tworzenie własnego startera

- Moduł startowy musi zawierać kod odpowiedzialny za auto-konfiguracje.
- Musimy dostraczyć odpowiednie zależności
- **@ConditionalOnClass**
- **@ConditionalOnExpression**
- **@ConditionalOnJava**
- **@ConditionalOnJndi**

@ConditionalOnMissingBean

- **@ConditionalOnMissingClass**
- **@ConditionalOnProperty**
- **@ConditionalOnResource**
- **@ConditionalOnWebApplication**
- **@ConditionalOnNotWebApplication**

19.9. Przykład

```
@Configuration
@ConditionalOnClass(JmsTemplate.class)
@ConditionalOnBean(ConnectionFactory.class)
@EnableConfigurationProperties(JmsProperties.class)
@AutoConfigureAfter({ HornetQAutoConfiguration.class,
ActiveMQAutoConfiguration.class })
public class JmsAutoConfiguration {
@Autowired
private JmsProperties properties;
@Autowired
private ConnectionFactory connectionFactory;
...

@Bean
@ConditionalOnMissingBean
public JmsTemplate jmsTemplate() {
JmsTemplate jmsTemplate = new
JmsTemplate(this.connectionFactory);
jmsTemplate.setPubSubDomain(this.properties.isPubSubDomain());
return jmsTemplate;
}
```

19.10. Health

-

```
@Component
public class ActiveMQHealth implements HealthIndicator {
private ConnectionFactory factory;
@Autowired
public ActiveMQHealth(ConnectionFactory factory) {
this.factory = factory;
}
@Override
public Health health() {
try {
factory.createConnection();
} catch (JMSException e) {
return new Health.Builder()
.down(e)
.build();
}
return new Health.Builder()
.status(Status.UP + ": Successfully connected to
the broker")
.build();
}
}
```

19.11. XML a Spring Boot

@ImportResource({"META-INF/services.xml","META-INF/repositories.xml"})

@ImportResource("classpath:applicationContext.xml")

20. Debug mode

```
debug=true
```

21. Hot swapping

- JRebel
- Spring Loaded

22. Banner

Podłącznie baneru będzie wykonywane automatycznie gdy plik **banner.txt, banner.gif, banner.jpg czy banner.png** znajdą się na classpath lub zostaną jawnie określone przez zmienną **banner.image.location**

${application.version} - wesja aplikacji zadeklarowana w MANIFEST.MF

```
Implementation-Version: 1.0 wyświetli 1.0.
```

${spring-boot.version} - wesja Spring Boot użytego w aplikacji

${application.title} - tytył aplikacji zadeklarowany w MANIFEST.MF.

```
Implementation-Title: MyApp wyślietli  MyApp.
```

SpringApplication.setBanner(...) Generowanie banneru programowalnie. (org.springframework.boot.Banner → printBanner() method)

```
spring:
    main:
        banner-mode: "off"
```

```
public static void main(String[] args) {
    SpringApplication app = new SpringApplication(MySpringConfiguration.class);
    app.setBannerMode(Banner.Mode.OFF);
    app.run(args);
}
```

22.1. Fluent builder API

```
new SpringApplicationBuilder()
    .bannerMode(Banner.Mode.OFF)
    .sources(Parent.class)
    .child(Application.class)
    .run(args);
```

23. Application events and listeners

ApplicationStartedEvent - odpalany podczas startu aplikacji, przed procesowaniem z wyjątkiem rejestracji słuchaczy czy initializerów.

ApplicationEnvironmentPreparedEvent - kiedy Environment zostanie użyty w kontekscie, ale przed stworzeniem kontekstu

ApplicationPreparedEvent - wywoływany przed odświeżeniem aplikacji ale po załadowaniu wszystkich definicji modułów.

ApplicationReadyEvent - wywoływany po odświeżeniu lub wywołaniu jakiegokolwiek wywołania

zwrotnego które zostało przeprocesowane w aplikacji

ApplicationFailedEvent - wywołany podczas wystąpienie wyjątku na starcie

24. Argumenty wywołania

```
@Component
public class MyBean {

    @Autowired
    public MyBean(ApplicationArguments args) {
        boolean debug = args.containsOption("debug");
        List<String> files = args.getNonOptionArgs();
        // if run with "--debug logfile.txt" debug=true, files=["logfile.txt"]
    }

}
```

25. Parametry z 'command line'

```
 java -jar -Dspring.profiles.active=production kataBoot-0.0.1-SNAPSHOT.jar
 java -jar -Dserver.port=8989 kataBoot-0.0.1-SNAPSHOT.jar
```

```
--server.port=9000
```

```
set SERVER_PORT=9999 // on windows machine
export SERVER_PORT=9999 // on linux
```

26. Admin features

spring.application.admin.enabled

27. Konfigurowanie zewnętrzne

- **@TestPropertySource - na teście**
- **Argumenty z linii poleceń - wywołanie**
- **Parametry z SPRING_APPLICATION_JSON**
- **Parametry zawarte w ServletConfig init**
- **Parametry zawarte w ServletContext init**

- **Skorzystanie z atrybutów JNDI poprzez java:comp/env.**
- **Skorzystanie z Java System properties (System.getProperties()).**
- **Skorzystanie z parametrów środowistkowych**
- **RandomValuePropertySource**
- **Profile-specific znajdujące się poza jarem (application-{profile}.properties & YAML)**
- **Profile-specific znajdujące się w środku jar (application-{profile}.properties & YAML)**
- **Application properties znajdujące się poza jar (application.properties & YAML)**
- **Application properties znajdujące się w środku jar (application.properties & YAML)**
- **@PropertySource na @Configuration**
- **Domyślne ustawienia poprzez użycie SpringApplication.setDefaultProperties**

application.properties & application.yml powinni znajdować się na classpath aplikacji

Zmienne środowiskowe sprawdzą się podczas praktyki z chmurami.

Zawsze możesz użyć command-line

```
data.server=remoteserver:3030
```

27.1. Zmiana lokacji pliku ustawień i jego nazwy

- /config umieszczony w bieżącym katalogu
- bieżący katalog
- /config w classpath
- classpath

Kolejność wyszukiwania jak wyżej

ConfigFileApplicationListener

```
private static final String DEFAULT_SEARCH_LOCATIONS =
"classpath:/,classpath:/config/,file:./,file:./config/";
```

27.2. Random

```
my.secret=${random.value}
my.number=${random.int}
my.bignumber=${random.long}
my.uuid=${random.uuid}
my.number.less.than.ten=${random.int(10)}
my.number.in.range=${random.int[1024,65536]}
```

27.3. Placeholders w pliku properties

```
app.name=MyApp
app.description=${app.name} is a Spring Boot application
```

27.4. Automatyczne wstawianie danych przez proces budujący → Maven

- Przykład

```
app.encoding=@project.build.sourceEncoding@
app.java.version=@java.version@
```

```
<resources>
    <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
    </resource>
</resources>


<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.7</version>
    <configuration>
        <delimiters>
            <delimiter>@</delimiter>
        </delimiters>
        <useDefaultDelimiters>false</useDefaultDelimiters>
    </configuration>
</plugin>
```

27.5. Automatyczne wstawianie danych przez proces budujący → Gradle

- Przykład

```
app.name=${name}
app.description=${description}
```

```
processResources {
    expand(project.properties)
}
```

27.6. YAML

SnakeYAML jest ładowany z automatu przez spring-boot-starter

```
environments:
    dev:
        url: http://dev.bar.com
        name: Developer Setup
    prod:
        url: http://foo.bar.com
        name: My Cool App
```

To samo co :

```
environments.dev.url=http://dev.bar.com
environments.dev.name=Developer Setup
environments.prod.url=http://foo.bar.com
environments.prod.name=My Cool App
```

```
my:
   servers:
       - dev.bar.com
       - foo.bar.com
```

To samo co :

```
my.servers[0]=dev.bar.com
my.servers[1]=foo.bar.com
```

27.7. ConfigurationProperties

- Przykład

```
@ConfigurationProperties(prefix="my")

@ConfigurationProperties(locations = "classpath:tomcat.properties",
ignoreUnknownFields = false, prefix = "tomcat")
```

27.8. Type-safe Configuration Properties

```
@ConfigurationProperties(prefix="connection")
public class ConnectionProperties {

    private String username;

    private InetAddress remoteAddress;

    // ... getters and setters

}
```

Zarejestrować :

```
@Configuration
@EnableConfigurationProperties(ConnectionProperties.class)
public class MyConfiguration {
}
```

```
@Component
@ConfigurationProperties(prefix="connection")
public class ConnectionProperties {

    // ... getters and setters
    private Resource location;  //connection.location=classpath:myfile.xml
    private boolean skip = true

}
```

```
# application.yml

connection:
    username: admin
    remoteAddress: 192.168.1.1

# additional configuration as required
```

Użycie:

```
@Service
public class MyService {

    private final ConnectionProperties connection;

    @Autowired
    public MyService(ConnectionProperties connection) {
        this.connection = connection;
    }

     //...

    @PostConstruct
    public void openConnection() {
        Server server = new Server();
        this.connection.configure(server);
    }

}
```

27.9. Relaxed binding

```
@ConfigurationProperties(prefix="person")
public class OwnerProperties {
    //Setter // Getter
    private String firstName;

}
```

Plik z właściwościami

```
person.firstName camel case

person.first-name .properties / .yml

person.first_name .properties / .yml

PERSON_FIRST_NAME Upper case format. system environment variables
```

@ConfigurationProperties nie powinnia działać dopóki nie ma @EnableConfigurationProperties na jednej z klas w projekcie.

Jeśli używasz SpringBoot :) to prawie na pewno, jakaś klasa pod spodem jest oznaczona tą adnotacją.

27.10. @ConfigurationProperties walidacje

JSR-303 javax.validation

```
@ConfigurationProperties(prefix="connection")
public class ConnectionProperties {

    @NotNull
    @Valid
    private InetAddress remoteAddress;

    // ... getters and setters

}
```

27.11. Meta-dane / Wsparcje dla IDE

Zależność:

```
compile('org.springframework.boot:spring-boot-configuration-processor')
```

27.12. Value

- @Value

```
@Value("${propertyName:defaultValue}")
```

```
java jar myapp.jar --propertyName=test
```

- Przykład

```
@PropertySource("classpath:/config.properties}")
@Configuration
```

lub

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

```
@Configuration
@ComponentScan(basePackages = { "pl.java.scalatech.setting" }, useDefaultFilters =
false, includeFilters = { @Filter(Component.class) })
@Slf4j
public class PropConfig {
    @Configuration
    @PropertySources({ @PropertySource(value = { "file:./conf/app.properties" }),
@PropertySource(value = { "file:./conf/app-dev.properties" }) })
    @Profile({ "dev", "dev_test"})
    static class PropertiesLoaderForDev {
        @Bean
        public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
            log.info("+++    dev profile launch");
            return new PropertySourcesPlaceholderConfigurer();
        }
    }

    @Configuration
    @PropertySources({ @PropertySource(value = { "file:./conf/app.properties" }),
@PropertySource(value = { "file:./conf/app-prod.properties" }) })
    @Profile("prod")
    static class PropertiesLoaderForProd {
        @Bean
        public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
            log.info("+++  prod profile launch");
            return new PropertySourcesPlaceholderConfigurer();
        }
    }

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource source = new ResourceBundleMessageSource();
        source.setBasename("i18n/messages");
        source.setUseCodeAsDefaultMessage(true);
        return source;
    }

}
```

27.13. PropertySource

Podanie dokładnej lokalizacji pliku z właściwościami

- @PropertySource
- Przykład

```
@PropertySource("file:${app.home}/app.properties")


@PropertySources({
    @PropertySource("classpath:tomcat.properties"),
    @PropertySource("classpath:db.properties")
})


@PropertySource(value="classpath:miss.properties", ignoreResourceNotFound=true)
```

28. Profile

- application-{profile}.properties
    - Przykład

```
application-qa.properties
server.ip=localhost

application-prod.properties
server.ip=http://my-remote.server.com

spring-boot:run -Dspring.profiles.active=prod
```

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

    // ...

}
```

```
spring.profiles.active=dev,hsqldb
```

28.1. Programowalne ustawianie profilu

SpringApplication.setAdditionalProfiles(...) & **ConfigurableEnvironment**

29. CommandLineRunner

```
public  void run(String...   args){
...
}
```

Zapewnia, że Spring Boot wykona daną metodę tylko raz zaraz po starcie

InitializingBean jako analogia

- Przykład

```
 @Autowired
    private Environment env;

    @Bean
    InitializingBean bean(){
        return () -> log.info("!!!!  application.user : {}",env.getProperty
(APPLICATION_USER));

    }
```

- Przykład 2 / @Autowired repository

```
@Bean
    InitializingBean populateDatabase(CarRepository repository) {
        return () -> {
            repository.save(new Car("Honda", "Civic", "red"));
            repository.save(new Car("Fiat", "Uno", "black"));
            repository.save(new Car("Ford", "Escort", "black"));
        };
}
```

30. ApplicationArguments

31. DevTools

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
    </dependency>
</dependencies>
```

```
dependencies {
    compile("org.springframework.boot:spring-boot-devtools")
}
```

```
spring.thymeleaf.cache
```

podobnie dla :

```
spring.thymeleaf.cache
spring.freemarker.cache
spring.velocity.cache
spring.mustache.cache
spring.groovy.template.cache
```

31.1. Tuning

```
spring:
devtools:
  restart:
     exclude: /static/**,/templates/**
```

31.2. LiveReload plugin

```
Wyłączenie : spring.devtools.livereload.enabled=false.
```

31.3. Uniemożliwienie restartów

```
public static void main(String[] args) {
    System.setProperty("spring.devtools.restart.enabled", "false");
    SpringApplication.run(MyApp.class, args);
}
```

31.4. Zdalne aplikacje

```
spring.devtools.remote.secret=mysecret

org.springframework.boot.devtools.RemoteSpringApplication
```

32. Spring loaded

Eclipse :

```
-javaagent:/home/przodownik/springloaded-1.2.6.RELEASE.jar -noverify
```

33. Logowanie

Spring Boot domyślnie używa Commons Logging dla wszystkich wewnętrznych logowań

Domyślna konfiguracja jest dostarczona dla rozwiązań typu :

- **Java Util Logging**
- **Log4J2**
- **Logback**

Domyślnie przez użycie 'Staters' Logback będzie użyty do logowania

Bardzo dobre rozwiązanie.

Połącz Logback z Lombok w celu eliminacji boilerplate code

```
Date and Time — Millisecond precision and easily sortable.
Log Level — ERROR, WARN, INFO, DEBUG or TRACE.
Process ID.
A --- separator to distinguish the start of actual log messages.
Thread name — Enclosed in square brackets (may be truncated for console output).
Logger name — This is usually the source class name (often abbreviated).
The log message.
```

Logback nie używa poziomu FATAL dla niego odpowiednikiem będzie ERROR

33.1. Console

```
java -jar myapp.jar --debug
```

```
debug=true in your application.properties.
```

```
--debug
```

33.2. Pliki z logami

Domyślnie Spring Boot loguje tylko na konsole a nie zapisuje logów do żadnych plików.

Jest to słabe rozwiązanie w wielu powodów.

33.3. Poziomy logowania

logging.level.root=WARN logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR

33.4. Konfiguracja logów użytkownika

- **Logback**

logback-spring.xml, logback-spring.groovy, logback.xml, logback.groovy

- **Log4j2**

log4j2-spring.xml, log4j2.xml

- **JDK (Java Util Logging)**

logging.properties

34. WEB

34.1. Aktualne kontenery

- **Tomcat 8 3.1 Java 7+**
- **Tomcat 7 3.0 Java 6+**
- **Jetty 9.3 3.1 Java 8+**
- **Jetty 9.2 3.1 Java 7+**
- **Jetty 8 3.0 Java 6+**
- **Undertow 1.3 3.1 Java 7+**

34.2. Spring MVC auto-configuration

- **Inicjalizacja DispatcherServlet'a**
- **Ustawienie 'encoding filter'**
- **Konfiguracja statycznych zasobów**
- **Włączenie ContentNegotiatingViewResolver & BeanViewResolver**
- **Wsparcie dla sewrowania kontentu statycznego & webjars**

- **Automatyczna rejestracja kowerterów i formaterów**
- **Wsparcie dla HttpMessageConvertes**
- **Automatyczna rejestracja MessageCodesResolver**
- **Wsparcie dla statycznego index.html**
- **Konfiguracja multipart resolver'a**
- **favicon**

34.3. Serwer port

```
server.port=9991
```

34.4. Static Content

```
spring.resources.staticLocations
```

- /META-INF/resources/
- /resources/
- /static/
- /public/

34.5. Template engines

- **FreeMarker**
- **Groovy**
- **Thymeleaf**
- **Velocity (deprecated in 1.4)**
- **Mustache**

34.6. favicon

Zasób:

```
src/main/resource/static/favicon.ico
```

HTML:

```
   <link rel="icon" type="image/x-icon" href="/favicon.ico">
```

35. Defiowanie filtrów

```
@Configuration
public class WebConfiguration{
    @Bean
    MyFilter myFilter(){
     return new MyFilter();
    }
}
```

Automatyczna detekcja : javax.servlet.Filter

36. Interceptory

- HandlerInterceptor

```
public class WebConfiguration extends WebMvcConfigurerAdapter   {  //krok 1

@Bean
LocaleChangeInterceptor localeChangeInterceptor() { //krok 2
        return new LocaleChangeInterceptor();
}

@Override
public void addInterceptors(InterceptorRegistry registry){ //krok 3
 registry.addInterceptor(localeChangeInterceptor());
}
}
```

Wywołanie :

```
http://{host}:{port}/{someUrl}?locale={locale_name}
```

37. PropertyEditors

```
public class CustomerEditor extends PropertyEditorSupport {
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
        if (StringUtils.hasText(text)) {
            setValue(new Customer(text.trim(), ""));
        } else {
            setValue(null);
        }
    }

    @Override
    public String getAsText() {
        Customer customer = (Customer) getValue();
        if (customer != null) {
            return customer.getFirstName();
        }
        return "";
    }
}
```

Dowiązanie :

```
@InitBinder
public  void initBinder(WebDataBinder binder)   {
  binder.registerCustomEditor(Customer.class, new CustomerEditor());
}
```

Kontroler :

```
@RequestMapping(value ="/customer/{firstName}", method = RequestMethod.GET)
public Customer getCustomer(@PathVariable Customer customer)
{
   return customer;
}
```

Wywołanie

```
http://{host}:{port}/person/przodownik
```

Nie jest to Thread-safe

38. Formatter

```
public class PersonFormatter implements  Formatter<Person> {
    private PersonRepository   repository;
    public  PersonFormatter(PersonRepository    repository) {
            this.repository =   repository;
    }
    @Override
    public  Person  parse(String  name, Locale  locale) throws   ParseException  {
            List<Person>  people  =  repository.findByFirstNameLike(name);
            return people.get(0);
    }

    @Override
    public  String  print(Person person,   Locale  locale) {
            return person.getFirstName();
    }
}
```

```
@Configuration
public class WebConfig extends WebMvcConfigurationSupport{
//...
@Override
public void addFormatters(FormatterRegistry registry)   {
  registry.addFormatter(new PersonFormatter(personRepo));
}
}
```

```
@RequestMapping(value ="/person/{login}", method = RequestMethod.GET)
public Person getPerson(@PathVariable Person person)
{
   return personRepository.findBookByLogin(person.getLogin());
}
```

39. Defiowanie ścieżek do własnych zasobów statycznych

```
@Configuration
public class WebConfig extends WebMvcConfigurationSupport{
//...
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry){
  registry.addResourceHandler("/resource/**").addResourceLocations("classpath:/");
}
```

- Przykład wywołania

```
http://{host}:{port}/resource/application.properties
```

40. Jetty zamiast Tomcat’a

Zależność Gradle :

```
compile("org.springframework.boot:spring-boot-starter-jetty")

configurations  {
  compile.exclude   module: "spring-boot-starter-tomcat"
}
```

41. Error Handling

@ControllerAdvice

- **@ExceptionHandler**
- **@ModelAttribute**
- **@InitBinder**

basePackageClasses - zawężamy skanowanie

- Przykład

```
@ControllerAdvice(basePackageClasses = FooController.class)
public class FooControllerAdvice extends ResponseEntityExceptionHandler {

    @ExceptionHandler(YourException.class)
    @ResponseBody
    ResponseEntity<?> handleControllerException(HttpServletRequest request, Throwable
ex) {
        HttpStatus status = getStatus(request);
        return new ResponseEntity<>(new CustomErrorType(status.value(), ex.getMessage
()), status);
    }

    private HttpStatus getStatus(HttpServletRequest request) {
        Integer statusCode = (Integer) request.getAttribute(
"javax.servlet.error.status_code");
        if (statusCode == null) {
            return HttpStatus.INTERNAL_SERVER_ERROR;
        }
        return HttpStatus.valueOf(statusCode);
    }

   @ExceptionHandler({ ObjectOptimisticLockingFailureException.class,
OptimisticLockingFailureException.class,
            DataIntegrityViolationException.class })
    @ResponseBody
    public ResponseEntity handleConflict(Exception ex) {
        return errorResponse(ex, HttpStatus.CONFLICT);
    }

 private ResponseEntity<ExceptionMessage> errorResponse(Throwable throwable,
HttpStatus status) {
        if (null != throwable) {
            log.error("error caught: " + throwable.getMessage(), throwable);
            return response(new ExceptionMessage(throwable), status);
        } else {
            log.error("unknown error caught in RESTController, {}", status);
            return response(null, status);
        }
}

 private <T> ResponseEntity<T> response(T body, HttpStatus status) {
        return new ResponseEntity<>(body, new HttpHeaders(), status);
}

}
```

- Przykład 2:

```
@ControllerAdvice
```

```
public class CustomResponseEntityExceptionHandler extends
ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid
(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status,
WebRequest request) {
        List<FieldError> fieldErrors = ex.getBindingResult().getFieldErrors();
        List<ObjectError> globalErrors = ex.getBindingResult().getGlobalErrors();
        List<String> errors = new ArrayList<>(fieldErrors.size() + globalErrors.size(
));
        String error;
        for (FieldError fieldError : fieldErrors) {
            error = fieldError.getField() + ", " + fieldError.getDefaultMessage();
            errors.add(error);
        }
        for (ObjectError objectError : globalErrors) {
            error = objectError.getObjectName() + ", " + objectError.
getDefaultMessage();
            errors.add(error);
        }
        ErrorMessage errorMessage = new ErrorMessage(errors);
        return new ResponseEntity(errorMessage, headers, status);
    }

    @Override
    protected ResponseEntity<Object> handleHttpMediaTypeNotSupported
(HttpMediaTypeNotSupportedException ex, HttpHeaders headers, HttpStatus status,
WebRequest request) {
        String unsupported = "Unsupported content type: " + ex.getContentType();
        String supported = "Supported content types: " + MediaType.toString(ex
.getSupportedMediaTypes());
        ErrorMessage errorMessage = new ErrorMessage(unsupported, supported);
        return new ResponseEntity(errorMessage, headers, status);
    }

    @Override
    protected ResponseEntity<Object> handleHttpMessageNotReadable
(HttpMessageNotReadableException ex, HttpHeaders headers, HttpStatus status,
WebRequest request) {
        Throwable mostSpecificCause = ex.getMostSpecificCause();
        ErrorMessage errorMessage;
        if (mostSpecificCause != null) {
            String exceptionName = mostSpecificCause.getClass().getName();
            String message = mostSpecificCause.getMessage();
            errorMessage = new ErrorMessage(exceptionName, message);
        } else {
            errorMessage = new ErrorMessage(ex.getMessage());
        }
        return new ResponseEntity(errorMessage, headers, status);
    }
```

```
}
```

Przykład 3:

```
@ControllerAdvice
public class GlobalControllerAdvice {

    @InitBinder
    public void configureBinding(WebDataBinder binder) {
        binder.setDisallowedFields("id");
    }

    @ResponseStatus(HttpStatus.CONFLICT)
    @ExceptionHandler(DataIntegrityViolationException.class)
    public void handleDatabaseConstraintViolation() {
      //..
    }

}
```

@ResponseStatus

```
@ResponseStatus(value = HttpStatus.NOT_FOUND, reason = "No such customer")
public class CustomerFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public CustomerFoundException(String key) {
        super(key + " not available");
    }
}
```

@ErrorHandler

- Przykład

```
    @ExceptionHandler({ NotFoundException.class })
    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ResponseBody
    public BadRequest handleException(NotFoundException e, HttpServletRequest request)
{
        log(String.format(LOG_RESPONSE_FORMAT, HttpStatus.NOT_FOUND.value(), request
.getRequestURI()), e.getMessage());
        return new BadRequest(e.getMessage());
    }


    @ExceptionHandler({ RuntimeException.class })
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ResponseBody
    public BadRequest handleException(RuntimeException e, HttpServletRequest request)
{
        log(String.format(LOG_RESPONSE_FORMAT, HttpStatus.INTERNAL_SERVER_ERROR.value
(), request.getRequestURI()), e.getMessage());
        return new BadRequest(e.toString());
    }

    @ExceptionHandler({ IllegalArgumentException.class })
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ResponseBody
    public BadRequest handleException(IllegalArgumentException e, HttpServletRequest
request) {
        log(String.format(LOG_RESPONSE_FORMAT, HttpStatus.BAD_REQUEST.value(),
request.getRequestURI()), e.getMessage());
        return new BadRequest(e.getMessage());
    }


    /**
     *  more informative message than a generic HTTP-400 Bad Request
     */
    @ExceptionHandler(HttpMessageNotReadableException.class)
    @ResponseBody
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public BadRequest handleException(HttpMessageNotReadableException e,
HttpServletRequest request) {
        ...
        return new BadRequest(e.getMessage());
}
```

```
@Configuration
public class ErrorConfig {


    @Bean
    public EmbeddedServletContainerCustomizer containerCustomizer() {
        return new MyCustomizer();
    }

    private static class MyCustomizer implements EmbeddedServletContainerCustomizer {

        @Override
        public void customize(ConfigurableEmbeddedServletContainer container) {
            container.addErrorPages(new ErrorPage(HttpStatus.BAD_REQUEST, "/400"));
        }

    }
}
```

@SimpleErrorHandler

@ResponseStatus

- Przykład

```
@ResponseStatus(value = HttpStatus.BAD_REQUEST)
public class BadRequest extends RuntimeException {
    public BadRequest(String message) {
        super(message);
    }
}

@ResponseStatus(HttpStatus.NOT_FOUND)
public static class NotFound extends RuntimeException {

}
}
```

42. SimpleMappingExceptionResolver

```
@Bean
   public SimpleMappingExceptionResolver createSimpleMappingExceptionResolver() {
     SimpleMappingExceptionResolver resolver = new SimpleMappingExceptionResolver();
     Properties errorMaps = new Properties();
     errorMaps.setProperty("MySpecificException", "error");
     errorMaps.setProperty("NullPointerException", "error");
     resolver.setExceptionMappings(errorMaps);
     resolver.setDefaultErrorView("globalerror");
     return resolver;
  }
```

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>My Exception Handling</title>
</head>
<body>

<h1>My Exception Handling</h1>

${exception.message}

</body>
</html>
```

42.1. Definiowanie stron błędów

Od Spring Boot 1.4 wystarczy stworzyć plik o danym kodzie błędu i umieścić do w zasobach statycznych. Automatyczne mapowanie.

- Przykład : 4..

```
src/
 +- main/
     +- java/
     |   + <source code>
     +- resources/
         +- public/
             +- error/
             |   +- 404.html
             +- <other public assets>
```

- Przykład : 5..

```
src/
 +- main/
     +- java/
     |   + <source code>
     +- resources/
         +- templates/
             +- error/
             |   +- 5xx.ftl
             +- <other templates>
```

43. Rest

43.1. Adnotacje

- **@Controller**
- **@RestController**
- **@ResponseBody**
- **@PathVariable**
- Przykład metody kontrolera

```
@RequestMapping("/user/{id}/{field}")
public void showUserField(@PathVariable("id") Long userId, @PathVariable("field")
String field) {
...
```

43.2. Prefix usługi

- Przykład

```
@Controller
@RequestMapping("/user")
public class UserController {
...
}
```

43.3. Spring Data REST

Zależność :

```
compile("org.springframework.boot:spring-boot-starter-data-rest")
```

- @RepositoryRestResource - oznaczenie danego repozytiorium

```
@RepositoryRestResource(collectionResourceRel="person",path="persons")
```

```
@RepositoryRestResource
public interface PersonRepo extends CrudRepository<Person, Long>
{
@RestResource(path="byName")
public List findByPersonName(@Param("name") String personName);
}
```

43.4. Wywoływanie serwisu REST

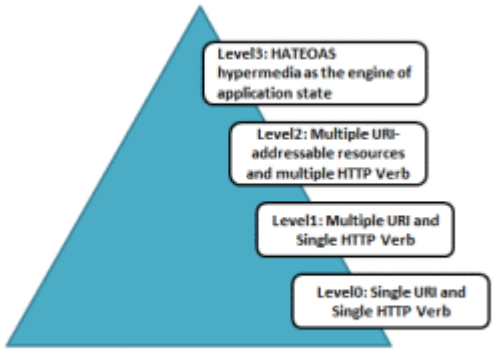
```
@Service
public class MyBean {

    private final RestTemplate restTemplate;

    public MyBean(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public Details someRestCall(String name) {
        return this.restTemplate.getForObject("/{name}/details", Details.class, name);
    }

}
```

43.5. Dostrojenie RESTTemplate

```
static class ProxyCustomizer implements RestTemplateCustomizer {

    @Override
    public void customize(RestTemplate restTemplate) {
        HttpHost proxy = new HttpHost("proxy.example.com");
        HttpClient httpClient = HttpClientBuilder.create()
                .setRoutePlanner(new DefaultProxyRoutePlanner(proxy) {

                    @Override
                    public HttpHost determineProxy(HttpHost target,
                            HttpRequest request, HttpContext context)
                                    throws HttpException {
                        if (target.getHostName().equals("192.168.0.5")) {
                            return null;
                        }
                        return super.determineProxy(target, request, context);
                    }

                }).build();
        restTemplate.setRequestFactory(
                new HttpComponentsClientHttpRequestFactory(httpClient));
    }

}
```

43.6. Richardson Maturity Model

source : https://technobeans.wordpress.com/



Richardson Maturity Model

43.7. Hal browser

Zależność :

```
compile('org.springframework.data:spring-data-rest-hal-browser')
```

http://{host}:{port}/browser/index.html

43.8. Swagger 2

Zależności :

```
compile group: 'io.springfox', name: 'springfox-swagger2', version: '2.5.0'
compile group: 'io.springfox', name: 'springfox-swagger-ui', version: '2.5.0'
```

Konfiguracja:

```
@Configuration
@EnableSwagger2
@Profile("rest")
public class SwaggerConfig extends WebMvcConfigurerAdapter{
    @Bean
    Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
          .select()
          .apis(RequestHandlerSelectors.any())
          .paths(PathSelectors.ant("/api/**"))
          .build()
          .apiInfo(apiInfo());
    }
    //TODO get info from properties

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title("Boot Kata - Web API")
            .description("Boot Kata based on REST")
            .license("Apache License Version 2.0")
            .version("1.0")
            .build();
    }
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("swagger-ui.html").addResourceLocations(
"classpath:/META-INF/resources/");
        registry.addResourceHandler("/webjars/**").addResourceLocations(
"classpath:/META-INF/resources/webjars/");
}
}
```

URL:

http://{host}:{port}/swagger-ui.html

lub REST

http://{host}:{port}/v2/api-docs/

43.9. Obsługa wyjątków

- **DefaultHandlerExceptionResolver**

```
@RestController
@RequestMapping("/customer")
public class CustomerController implements HandlerExceptionResolver {

    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<Object> get(){
        throw new RuntimeException("customer not yet supported");
    }

    @Override
    public ModelAndView resolveException(HttpServletRequest req, HttpServletResponse
resp, Object handler, Exception ex) {
        resp.reset();
        resp.setCharacterEncoding("UTF-8");
        resp.setContentType("text/json");

        ModelAndView model = new ModelAndView(new MappingJackson2JsonView());
        if (ex instanceof RuntimeException){
            resp.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            model.addObject("code", HttpStatus.INTERNAL_SERVER_ERROR.value());
            model.addObject("message", ex.getMessage());
        } else {
            resp.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            model.addObject("code", HttpStatus.INTERNAL_SERVER_ERROR.value());
            model.addObject("message", ex.getMessage());
        }
        return model;
    }
}
```

- **@ControllerAdvice**

```
Przykład powyżej
```

- **@ExceptionHandler**

```
Przykład powyżej
```

- **@ResponseStatus**

```
Przykład powyżej
```

- **ErrorMvcAutoConfiguration**

```
ErrorController

${error.path:/error}

${error.whitelabel.enabled:true}
```

- **SimpleMappingExceptionResolver**

```
Przykład powyżej
```

43.10. Hateoas

- **Resource, Resources<T>**

```
public class ProductResource extends ResourceSupport {

    private UUID uuid;
    private String name;
    private String supplier;
    private Double price;

    @JsonCreator
    public ProductResource(@JsonProperty(value = "uuid") @NotNull UUID uuid,
                           @JsonProperty(value = "name") @NotNull String name,
                           @JsonProperty(value = "supplier") @NotNull String supplier,
                           @JsonProperty(value = "price") @NotNull Double price) {
        this.uuid = uuid;
        this.name = name;
        this.supplier = supplier;
        this.price = price;
}
```

- **ResourceAssembler**

```
public class ProductResourceAssembler extends ResourceAssemblerSupport<Product,
ProductResource> {

    public ProductResourceAssembler() {
        super(ProductController.class, ProductResource.class);
    }

    @Override
    public ProductResource toResource(Product product) {

        ProductResource resource = createResourceWithId(product.getUuid(), product);
        BeanUtils.copyProperties(product, resource);

        return resource;
    }

    public Product toClass(ProductResource resource){

        return new Product(resource.getUuid(), resource.getName(), resource
.getSupplier(), resource.getPrice());

    }
}
```

```
final class OwnerResourceAssembler implements ResourceAssembler<Owner,
Resource<Owner>> {

    @Override
    public Resource<Owner> toResource(Owner owner) {
        Cat cat = new Cat();
        Resource<Owner> resource = new Resource<>(owner);

resource.add(linkTo(methodOn(HateoasOwnerController.class).read(owner.getId())).withSe
lfRel());
        resource.add(linkTo(methodOn(HateoasCatController.class,
owner.getId()).read(cat.getId())).withRel("cat"));

        return resource;
    }

}
```

- **PagedResourcesAssembler**

```
  @GetMapping(path = "/")
    PagedResources<List<Resource<Customer>>> getAll(Pageable pageable) {
        Page<Customer> result = customerRepository.findAll(pageable);
        List<Resource<Customer>> customers = result.getContent().stream().map(this:
:toResource).collect(toList());
        return new PagedResources(customers, getMetadata(result), getLinks(result));

    }

    @GetMapping(path="/second",produces = MediaType.APPLICATION_JSON_VALUE)
    public HttpEntity<PagedResources<Customer>> page(Pageable pageable
,PagedResourcesAssembler pagedResourcesAssembler) {
        return ResponseEntity.ok((pagedResourcesAssembler.toResource
(customerRepository.findAll(pageable),customerAssembler )));
}

    @Autowired
    private UserResourceAssembler resourceAssembler;

    @RequestMapping(method = RequestMethod.GET, produces = MediaType
.APPLICATION_JSON_VALUE)
    public PagedResources<User> getUsers(Pageable pageable, PagedResourcesAssembler
assembler) {
        return assembler.toResource(userService.getUsers(pageable), resourceAssembler
);
}
```

```
 @RequestMapping(method = GET)
    public Callable<HttpEntity<?>> getUsers(Pageable pageable,
PagedResourcesAssembler<User> pagedResourcesAssembler) {
        return () -> {
            List<User> users = repository.findAll(pageable);
            Page<User> page = new PageImpl<>(users, pageable, repository.count());
            PagedResources<Resource<User>> resources =
pagedResourcesAssembler.toResource(page, resourceAssembler());
            resources.add(linkTo(methodOn(UserController.class).users(null,
null)).withSelfRel());

resources.add(linkTo(methodOn(UserController.class).invoice(null)).withRel("invoice"))
;
            return ok(resources);
        };
}
```

- **links (self)**

```
   @GetMapping(path = "/all")
    ResponseEntity<Resources<Resource<Customer>>> getAll() {
        List<Link> links = new LinkedList<>();
        links.add(linkTo(methodOn(getClass()).getAll()).withSelfRel());
        List<Resource<Customer>> customers = customerRepository.findAll().stream().
map(this::toResource).collect(Collectors.toList());
        return ResponseEntity.ok(new Resources<>(customers, links));
    }

    private Resource<Customer> toResource(final Customer customer) {
        List<Link> links = new LinkedList<>();
        links.add(linkTo(methodOn(CustomerController.class).getCustomereByID(customer
.getId())).withSelfRel());
        return new Resource<>(customer, links);
    }

    private Resource<Customer> toResource(final Customer customer) {
        List<Link> links = new LinkedList<>();
        links.add(linkTo(methodOn(CustomerController.class).getCustomereByID(customer
.getId())).withSelfRel());
        return new Resource<>(customer, links);
    }
```

- **ControllerLinkBuilder**

```
public class TaskResource extends Resource<Task> {

    TaskResource(Task task) {
        super(task);

       add(ControllerLinkBuilder.linkTo(ControllerLinkBuilder.methodOn(TaskController
.class).read(task.getId()))
                .withSelfRel());

    add(ControllerLinkBuilder.linkTo(ControllerLinkBuilder.methodOn(TaskController
.class).edit(task.getId(), new Task()))
                .withRel("tasks"));

    add(ControllerLinkBuilder.linkTo(ControllerLinkBuilder.methodOn(TaskController
.class).list())
                .withRel("previous"));

    }

}
```

```
public class CityResource extends Resource<City> {

    public CityResource(City entity) {
        super(entity, linkTo(methodOn(CityHateoasController.class).item(entity.getId(
))).withSelfRel(), linkTo(
                CityHateoasController.class).slash(entity.getId()).withRel("delete"));
    }
}
```

44. JSON

44.1. @JsonView

Domain class:

```
public class User extends AbstractEntity{

    public static interface OnlyLoginView {}
    public static interface OnlyEmailView {}
    public static interface OnlyEnabledView {}
    public static interface OnlyVersionView {}
    public static interface AllView extends OnlyLoginView, OnlyEmailView,
OnlyEnabledView, OnlyVersionView {}

    private static final long serialVersionUID = -8920961125119379475L;
    private  String firstname;
    @JsonView(User.OnlyEmailView.class)
    private  String email;
    @JsonView(User.OnlyLoginView.class)
    private String login;
    private String password;
    @JsonView(User.OnlyEnabledView.class)
    private boolean enabled;
    @Transient
    private LocalDate birthDate;

    private String lastName;

    @Enumerated
    private Gender gender;
```

Controller class :

```
@JsonView(User.OnlyEmailView.class)
public User getUser(@PathVariable("userId") Long userId) {
        Meter requests = metricRegistry.meter("requestsId");
        requests.mark();
        return userRepository.findOne(userId);

}
```

Przykład 2 :

```
@RestController
public class UserController {

    @GetMapping("/user")
    @JsonView(User.WithoutPasswordView.class)
    public User getUser() {
        return new User("przodownik", "!@@vve3223");
    }
}


@Data
//..
public class User {

    public interface WithoutPasswordView {};
    public interface WithPasswordView extends WithoutPasswordView {};

    @JsonView(WithoutPasswordView.class)
    private String username;

   @JsonView(WithPasswordView.class)
    private String password;


}
```

@JsonIgnore

XStream ⇒ @XstreamOmitField

44.2. @JsonBackReference

Serializacja wstecz pominięta

44.3. @JsonManagedReference

Normalna serializacja wstecz

```
@RunWith(SpringRunner.class)
@JsonTest
@Slf4j
public class JsonSimpleTest {
    @Autowired
    private ObjectMapper objectMapper;

    @Test
    public void shouldSerializationTest() throws JsonProcessingException{
        User user = new User(1, "Przodownik",newArrayList());
        Item i1 = new Item(1, "tv", user);
        Item i2 = new Item(2, "bike", user);
        user.addItem(i1);
        user.addItem(i2);

       log.info("user: {}",objectMapper.writeValueAsString(user));
       log.info("item1 : {}",objectMapper.writeValueAsString(i1));
    }
}
```

44.4. @JsonPropertyOrder

```
@Data
@JsonPropertyOrder({ "name", "age","lastName","addressList" })
public class Person {

    private String name;
    private String lastName;
    private int age;
    private List<Address> addressList;
```

44.5. @JsonSerialize

```
public class Alert {
    public String name;
    //...
    @JsonSerialize(using = MyDateSerializer.class)
    public Date alertDate;
}
```

```
public class MyDateSerializer extends StdSerializer<Date> {

    private static SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy
hh:mm:ss");

    public MyDateSerializer() {
        this(null);
    }

    public MyDateSerializer(Class<Date> t) {
        super(t);
    }

    @Override
    public void serialize(Date value, JsonGenerator gen, SerializerProvider arg2)
      throws IOException, JsonProcessingException {
        gen.writeString(formatter.format(value));
    }
}
```

44.6. @JsonCreator - serializacja

```
public class CreatorJsonExample {
    public int id;
    public String name;

    @JsonCreator
    public CreatorJsonExample(@JsonProperty("name") String name, @JsonProperty("id")
int id) {
        this.id = id;
        this.name = name;
    }
}
```

Test:

```
@RunWith(SpringRunner.class)
@JsonTest
@Slf4j
public class JsonSimpleTest {
    @Autowired
    private ObjectMapper objectMapper;
    @Test
    public void shouldCreateJsonWork() throws JsonProcessingException, IOException {

        String json = "{\"id\":2,\"name\":\"przodownik\"}";
        CreatorJsonExample bean =
objectMapper.readerFor(CreatorJsonExample.class).readValue(json);
        assertEquals("przodownik", bean.name);
    }
```

45. SSL

45.1. Generacja klucza

```
keytool -genkey -alias springboot -keyalg RSA -keystore
src/main/resources/tomcat.keystore
```

45.2. Ustawienia

```
server.port = 8443
server.ssl.key-store = classpath:keystore.jks
server.ssl.key-store-password = secret
server.ssl.key-password = another-secret
```

```
@Configuration
public class SecureTomcatConfiguration {
@Bean
public EmbeddedServletContainerFactory servletContainer() throws FileNotFoundException
{
TomcatEmbeddedServletContainerFactory f = new TomcatEmbeddedServletContainerFactory();
f.addAdditionalTomcatConnectors(createSslConnector());
return f;
}
private Connector createSslConnector() throws
FileNotFoundException { Connector connector = new Connector(Http11NioProtocol.class
.getName());
Http11NioProtocol protocol = (Http11NioProtocol)connector.getProtocolHandler();
connector.setPort(8443);
connector.setSecure(true);
connector.setScheme("https");
protocol.setSSLEnabled(true);
protocol.setKeyAlias("springboot");
protocol.setKeystorePass("password");
protocol.setKeystoreFile(ResourceUtils.getFile("src/main/resources/tomcat.keystore").g
etAbsolutePath());
protocol.setSslProtocol("TLS");
return connector;
}
}
```

```
  @Bean
    public EmbeddedServletContainerFactory servletContainer() {
        final TomcatEmbeddedServletContainerFactory tomcatFactory = new
TomcatEmbeddedServletContainerFactory();

        // tomcatFactory.setAddress(InetAddress.getLocalHost());// you can restrict
localhost access
        tomcatFactory.setPort(8080);
        // ServletContainerInitializer

        /** This line has no significance . handled by the addViewControllers() method
*/
        tomcatFactory.addContextCustomizers(context ->
context.addWelcomeFile("/api/swagger-ui.html"));

        try {
            final ClassPathResource classPathResource = new
ClassPathResource("keystore");

            /** This code snippet enabled SSL . when you run from executable Jar this
is not working. adding if condition to avoid error */
            if (classPathResource.getFile().exists()) {
                final Connector connector = new Connector();
                connector.setPort(8443);
                connector.setSecure(true);
                connector.setScheme("https");
                connector.setProperty("SSLEnabled", "true");
                connector.setProperty("keystorePass", "spring");
                connector.setProperty("keystoreFile",
classPathResource.getFile().getAbsolutePath());
                tomcatFactory.addAdditionalTomcatConnectors(connector);
            }

        } catch (final Exception e) {
            LOGGER.debug("Error while loading classpath resource  ", e);
        }

        return tomcatFactory;
    }
```

```
@Bean
public EmbeddedServletContainerFactory servletContainer() {
    TomcatEmbeddedServletContainerFactory factory = new
TomcatEmbeddedServletContainerFactory();
    factory.setPort(9000);
    factory.setSessionTimeout(10, TimeUnit.MINUTES);
    factory.addErrorPages(new ErrorPage(HttpStatus.NOT_FOUND, "/notfound.html"));
    return factory;
}
```

46. Context path

```
server.contextPath=/kataSpringboot
```

Sposób programistyczny

```
@Component
public class CustomizationBean implements EmbeddedServletContainerCustomizer {
    @Override
    public void customize(ConfigurableEmbeddedServletContainer container) {
        container.setContextPath("/kataSpringboot");
    }
}
```

47. Dalsza optymalizacja

47.1. Zasoby statyczne :

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
                .addResourceLocations("/public-resources/")
                .setCacheControl(CacheControl.maxAge(1,
TimeUnit.HOURS).cachePublic());
    }

}
```

47.2. Cache-Control, ETag, Last-Modified

```
@GetMapping("/book/{id}")
public ResponseEntity<Book> showBook(@PathVariable Long id) {

    Book book = findBook(id);
    String version = book.getVersion();

    return ResponseEntity
                .ok()
                .cacheControl(CacheControl.maxAge(30, TimeUnit.DAYS))
                .eTag(version) // lastModified is also available
                .body(book);
}
```

Aktywacja:

```
@Bean
public Filter etagFilter() {
  return new ShallowEtagHeaderFilter();
}
```

Przykład 2:

```
 @RequestMapping(value = "/employees/{id}", method = RequestMethod.GET)
    public ResponseEntity<Customer> findOne(@PathVariable("id") long id) {
        Customer customer = customerService.findOne(id);
        return ResponseEntity
                .ok()
                .eTag(hashToEtag(customer))
                .body(customer);
    }


private String hashToEtag(Object o) {
        return "\"" + Integer.toString(o.hashCode()) + "\"";
}
```

47.3. Parametryzowanie:

```
spring.thymeleaf.cache=true
spring.messages.cache-seconds=-1

security.headers.cache=false

#3 dni
spring.resources.cache-period=259200
```

47.4. Gzip

```
server.tomcat.compression=on
server.tomcat.compressableMimeTypes=text/html,text/xml,text/css,text/plain,application
/json,application/xml,application/javascript
```

47.5. Cache + @Cacheable

```
@Configuration
@EnableCaching
public class CacheConfiguration {
 @Bean
  public CacheManager cacheManager() {
     SimpleCacheManager simpleCacheManager = new SimpleCacheManager();
     simpleCacheManager.setCaches(Arrays.asList(new ConcurrentMapCache("customers")));
     return simpleCacheManager;
 }
}
```

47.6. Cache Guava

```
@Configuration
@EnableCaching
public class CacheConfiguration {
@Bean
public CacheManager cacheManager() {
  GuavaCacheManager cacheManager = new GuavaCacheManager("customers");

cacheManager.setCacheBuilder(CacheBuilder.newBuilder().softValues().expireAfterWrite(2
0, TimeUnit.MINUTES));
return cacheManager;
}
}
```

48. ORM

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.hsqldb</groupId>
    <artifactId>hsqldb</artifactId>
    <scope>runtime</scope>
</dependency>
```

- Przykład z H2

```
dependencies {
compile("org.springframework.boot:spring-boot-starter-data-jpa")
compile("com.h2database:h2")
}
```

Jeśli używasz H2 i devTools to domyślnie aktywuje się konsola webowa /h2-console

NOTE : spring-jdbc - dla wbudowanej bazy auto-konfiguracja

NOTE : Jeśli chcesz używać instancji bazy nawet po wyłączeniu aplikacji ustaw : DB_CLOSE_ON_EXIT=FALSE

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

- Przykład programistycznego tworzenia dataSource

```
@Bean
public DataSource dataSource() {
DriverManagerDataSource dataSource = new DriverManagerDataSource();
dataSource.setDriverClassName("com.mysql.jdbc.Driver");
dataSource.setUrl("jdbc:mysql://localhost:3306/db1");
dataSource.setUsername("user1");
dataSource.setPassword("pass1");
return dataSource;
}
```

Domyślne ustawienia:

```
# Number of ms to wait before throwing an exception if no connection is available.
spring.datasource.tomcat.max-wait=10000

# Maximum number of active connections that can be allocated from this pool at the
same time.
spring.datasource.tomcat.max-active=50

# Validate the connection before borrowing it from the pool.
spring.datasource.tomcat.test-on-borrow=true
```

48.1. Ładowanie danych z skryptu SQL

```
spring.datasource.platform: h2
```

Spring szuka sktyptu **data.sql** oraz dla innej platformy **db data-${platform}.sql**

```
src/main/resources/data-h2.sql

insert into Person
(id, name)
values
(1, 'przodownik');
```

48.2. Ładowanie danych programowe poprzez repozytoria

@PostConstruct lub @CommandLineRunner

48.3. www.liquibase.org[Liquibase]

Gradle

```
compile("org.liquibase:liquibase-core")
```

Maven

```
<dependency>
<groupId>org.liquibase</groupId>
<artifactId>liquibase-core</artifactId>
</dependency>
```

48.4. Flyway

Gradle

```
compile("org.flywaydb:flyway-core")
```

Maven

```
<dependency>
<groupId>org.flywayfb</groupId>
<artifactId>flyway-core</artifactId>
</dependency>
```

48.5. JNDI

```
spring.datasource.jndi-name=java:jboss/datasources/customers
```

48.6. JdbcTemplate

```
@Component
public class MyBean {

    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public MyBean(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // ...

}
```

- Tradycyjne podejście

```
@Bean
public JdbcTemplate jdbcTemplate(DataSource dataSource) {
return new JdbcTemplate(dataSource);
}

@Bean
public DataSource dataSource() {
 return new
EmbeddedDatabaseBuilder().setType(EmbeddedDatabaseType.H2).addScripts('schema.sql',
'data.sql').build();
}
```

- Podejście Spring Boot

Jeśli Spring Boot wykryje bibiotekę H2 w swoim classpath automatycznie skonfiguruje wbudowanę bazę. Jeśli dodatkowo wykryje JdbcTemplate w swoim classpath skonfiguruje też tę zależność za Ciebie.

48.7. JPA & Spring Data

```
**Hibernate** — popularna implementacja JPA
**Spring Data JPA** — proste tworzenie repozytoriów
**Spring ORMs** — Core ORM wspieranie ORM'a przez Spring
```

48.8. Użycie H2’s web console

Przykład

```
    com.h2database:h2 is on the classpath
```

Ustawienie :

```
spring.h2.console.enabled
```

Ścieżka :

```
/h2-console
```

48.9. Zabezpieczenie h2-console

```
    security.user.role
    security.basic.authorize-mode
    security.basic.enabled
```

49. Źródła wiedzy

- Hibernate in Action
- Java Persistence with Hibernate
- Java JEE 6
- Pro JPA 2
- Pro JPA 2: Mastering the Java™ Persistence API (Expert's Voice in Java Technology)
- Hibernate from Novice to Professional
- Spring Data Modern Data Access for Enterprise Java
- Spring Data
- Spring Boot
- Spring Essentials
- Spring in Action
- etc

50. Spring Data / Hibernate / JPA

50.1. Najnowsze features:)

- Support for Projections in repository query methods.
- Support for Query by Example.
- The following annotations have been enabled to build own, composed annotations: @EntityGraph, @Lock, @Modifying, @Query, @QueryHints and @Procedure.
- Support for Contains keyword on collection expressions.
- AttributeConverters for ZoneId of JSR-310 and ThreeTenBP.
- Upgrade to Querydsl 4, Hibernate 5, OpenJPA 2.4 and EclipseLink 2.6.1.

50.2. Geneza

```
Powstał 2010 podczas sesji Roda Jonsona z Emilem Eifrem (Neo Technologies), jako próba
integracji Spring z Neo4j
```

50.3. Charakretystyka

- generyczna impl DAO
- postawowa impl CRUD
- kod dao generowany poprzez definicję metody np findByName , findByAgeBetween
- paging i sortowanie w standardzie
- wsparcie dla JPA, Mongo, Neo4j, Redis itd
- wsparcie dla QueryDSL

source : *www.patrick-gotthard.de*

[repository inheritance hierarchy] | *repository-inheritance-hierarchy.png*

source: *https://visola.github.io*

[spring data 001] | *spring-data-001.png*

source: *http://luxosuseadicto.blogspot.com*

[dataaccess jpa basic flow] | *dataaccess_jpa_basic_flow.png*

50.4. Narzędzia

— http://start.spring.io/

— Spring boot console : spring init -dweb,data-jpa,h2,thymeleaf --build gradle simpleApp --force

— STS

50.4.1. Przykład całej minimalnej konfiguracji

- Przykład

```
@Configuration
@EnableJpaRepositories
@EnableTransactionManagement
class ApplicationConfig {

  @Bean
  public DataSource dataSource() {

    EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
    return builder.setType(EmbeddedDatabaseType.HSQL).build();
  }

  @Bean
  public EntityManagerFactory entityManagerFactory() {

    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    vendorAdapter.setGenerateDdl(true);

    LocalContainerEntityManagerFactoryBean factory = new
LocalContainerEntityManagerFactoryBean();
    factory.setJpaVendorAdapter(vendorAdapter);
    factory.setPackagesToScan("com.acme.domain");
    factory.setDataSource(dataSource());
    factory.afterPropertiesSet();

    return factory.getObject();
  }

  @Bean
  public PlatformTransactionManager transactionManager() {

    JpaTransactionManager txManager = new JpaTransactionManager();
    txManager.setEntityManagerFactory(entityManagerFactory());
    return txManager;
  }
}
```

50.4.2. Tworzenie proxy

- Przykład

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@EnableJpaRepositories
class Config {}
```

50.4.3. Annotation-driven configuration

- Przykład

```
@EnableJpaRepositories(basePackages = "com.acme.repositories.jpa")
@EnableMongoRepositories(basePackages = "com.acme.repositories.mongo")
interface Configuration { }
```

50.4.4. Standalone

- Przykład

```
RepositoryFactorySupport factory = ⋯ // Instantiate factory here
UserRepository repository = factory.getRepository(UserRepository.class);
```

50.4.5. Wybór metod CRUD

- Przykład

```
@NoRepositoryBean
interface MyBaseRepository<T, ID extends Serializable> extends Repository<T, ID> {

  T findOne(ID id);

  T save(T entity);
}

interface UserRepository extends MyBaseRepository<User, Long> {
  User findByEmailAddress(EmailAddress emailAddress);
}
```

50.4.6. Wykorzystanie Pageable, Slice, Sort

- Przykład

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

50.4.7. Ograniczenie wyników zapytań

- Przykład

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

50.4.8. Streaming

- Przykład

```
@Query("select u from User u")
Stream<User> findAllByCustomQueryAndStream();

Stream<User> readAllByFirstnameNotNull();

@Query("select u from User u")
Stream<User> streamAllPaged(Pageable pageable);

///


try (Stream<User> stream = repository.findAllByCustomQueryAndStream()) {
  stream.forEach(…);
}
```

50.4.9. Asynchroniczność

- Przykład

```
@Async
Future<User> findByFirstname(String firstname);

@Async
CompletableFuture<User> findOneByFirstname(String firstname);

@Async
ListenableFuture<User> findOneByLastname(String lastname);
```

50.4.10. Dostrajanie do swoich potrzeb

- Przykład

```
interface UserRepositoryCustom {
  public void someCustomMethod(User user);
}



class UserRepositoryImpl implements UserRepositoryCustom {

  public void someCustomMethod(User user) {
    // Your custom implementation
  }
}




interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustom {

  // Declare query methods here
}
```

50.4.11. DSL

- Przykład

```
public interface QueryDslPredicateExecutor<T> {

    T findOne(Predicate predicate);

    Iterable<T> findAll(Predicate predicate);

    long count(Predicate predicate);

    boolean exists(Predicate predicate);

    // … more functionality omitted.



interface UserRepository extends CrudRepository<User, Long>,
QueryDslPredicateExecutor<User> {

}


}
```

- Przykład

```
Predicate predicate = user.firstname.equalsIgnoreCase("dave")
    .and(user.lastname.startsWithIgnoreCase("mathews"));

userRepository.findAll(predicate);
```

50.5. Nazwane zapytania

- Przykład

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
  query = "select u from User u where u.emailAddress = ?1")
public class User {

}



public interface UserRepository extends JpaRepository<User, Long> {

  List<User> findByLastname(String lastname);

  User findByEmailAddress(String emailAddress);
}
```

50.5.1. @Query

- Przykład

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}



public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.firstname like %?1")
  List<User> findByFirstnameEndsWith(String firstname);
}
```

50.5.2. Natywne zapytania

- Przykład

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
  User findByEmailAddress(String emailAddress);
}

public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.firstname = :firstname or u.lastname =
:lastname")
  User findByLastnameOrFirstname(@Param("lastname") String lastname,
                                 @Param("firstname") String firstname);
}
```

50.5.3. SpEL expressions

- Przykład

```
@Entity
public class User {

  @Id
  @GeneratedValue
  Long id;

  String lastname;
}

public interface UserRepository extends JpaRepository<User,Long> {

  @Query("select u from #{#entityName} u where u.lastname = ?1")
  List<User> findByLastname(String lastname);
}
```

50.5.4. Modyfikacja danych

```
@Modifying
@Query("update User u set u.firstname = ?1 where u.lastname = ?2")
int setFixedFirstnameFor(String firstname, String lastname);
```

50.5.5. Hint

- Przykład

```
public interface UserRepository extends Repository<User, Long> {

  @QueryHints(value = { @QueryHint(name = "name", value = "value")},
              forCounting = false)
  Page<User> findByLastname(String lastname, Pageable pageable);
}
```

50.5.6. Fetch load EntityGraph

- Przykład

```
@Entity
@NamedEntityGraph(name = "GroupInfo.detail",
  attributeNodes = @NamedAttributeNode("members"))
public class GroupInfo {

  // default fetch mode is lazy.
  @ManyToMany
  List<GroupMember> members = new ArrayList<GroupMember>();

  ...

}
@Repository
public interface GroupRepository extends CrudRepository<GroupInfo, String> {

  @EntityGraph(value = "GroupInfo.detail", type = EntityGraphType.LOAD)
  GroupInfo getByGroupName(String name);

}



@Repository
public interface GroupRepository extends CrudRepository<GroupInfo, String> {

  @EntityGraph(attributePaths = { "members" })
  GroupInfo getByGroupName(String name);

}
```

50.5.7. Projection

- Przykład

```
@Entity
public class Person {

  @Id @GeneratedValue
  private Long id;
  private String firstName, lastName;

  @OneToOne
  private Address address;
  …
}

@Entity
public class Address {

  @Id @GeneratedValue
  private Long id;
  private String street, state, country;

  …
}


interface PersonRepository extends CrudRepository<Person, Long> {

  Person findPersonByFirstName(String firstName);
}



interface AddressRepository extends CrudRepository<Address, Long> {}



interface NoAddresses {

  String getFirstName();

  String getLastName();
}
```

50.5.8. Procedury składowane

- Przykład

```
/;
DROP procedure IF EXISTS plus1inout
/;
CREATE procedure plus1inout (IN arg int, OUT res int)
BEGIN ATOMIC
 set res = arg + 1;
END
/;
```

```
@Entity
@NamedStoredProcedureQuery(name = "User.plus1", procedureName = "plus1inout",
parameters = {
  @StoredProcedureParameter(mode = ParameterMode.IN, name = "arg", type = Integer
.class),
  @StoredProcedureParameter(mode = ParameterMode.OUT, name = "res", type = Integer
.class) })
public class User {}


@Procedure("plus1inout")
Integer explicitlyNamedPlus1inout(Integer arg);



@Procedure(procedureName = "plus1inout")
Integer plus1inout(Integer arg);



@Procedure(name = "User.plus1IO")
Integer entityAnnotatedCustomNamedProcedurePlus1IO(@Param("arg") Integer arg);




@Procedure
Integer plus1(@Param("arg") Integer arg);
```

50.5.9. Specifications

- Przykład

```
public interface CustomerRepository extends CrudRepository<Customer, Long>,
JpaSpecificationExecutor {
 ...
}



List<T> findAll(Specification<T> spec);



public interface Specification<T> {
  Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
            CriteriaBuilder builder);
}



public class CustomerSpecs {

  public static Specification<Customer> isLongTermCustomer() {
    return new Specification<Customer>() {
      public Predicate toPredicate(Root<Customer> root, CriteriaQuery<?> query,
            CriteriaBuilder builder) {

         LocalDate date = new LocalDate().minusYears(2);
         return builder.lessThan(root.get(_Customer.createdAt), date);
      }
    };
  }

  public static Specification<Customer> hasSalesOfMoreThan(MontaryAmount value) {
    return new Specification<Customer>() {
      public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query,
            CriteriaBuilder builder) {

         // build query here
      }
    };
  }
}

// using


List<Customer> customers = customerRepository.findAll(isLongTermCustomer());
```

- Przykład 2

```
public class UserSpecifications {

    public static Specification<User> getUserByLogin(final String str) {
        return new Specification<User>() {
            @Override
            public Predicate toPredicate(Root<User> personRoot, CriteriaQuery<?>
query, CriteriaBuilder cb) {
                return cb.equal(personRoot.<String> get(User_.login), str);
            }
        };
    }

    public static Specification<User> getUsersWhoEarMoreThan(final BigDecimal salary)
{
        return new Specification<User>() {
            @Override
            public Predicate toPredicate(Root<User> personRoot, CriteriaQuery<?>
query, CriteriaBuilder cb) {
                return cb.greaterThan(personRoot.<BigDecimal> get(User_.salary),
salary);
            }
        };
    }
}

//
```

- Przykład

```
@Test
    public void shouldSpecificationsPredicateWork() {
        assertThat(userRepository.findAll(getUserByLogin("przodownik"))).hasSize(1)
            .containsOnly(User.builder().login("przodownik").name("borowiec").salary
(new BigDecimal(120)).build());
        assertThat(userRepository.findAll(getUsersWhoEarMoreThan(new BigDecimal(300))
)).hasSize(2);
    }
```

50.5.10. Query by Example

```
*** Przykład
```

```
public class Person {

  @Id
  private String id;
  private String firstname;
  private String lastname;
  private Address address;

  // … getters and setters omitted
}

Person person = new Person();
person.setFirstname("Dave");

Example<Person> example = Example.of(person);

public interface QueryByExampleExecutor<T> {

  <S extends T> S findOne(Example<S> example);

  <S extends T> Iterable<S> findAll(Example<S> example);

  // … more functionality omitted.
}



//example

Person person = new Person();
person.setFirstname("Dave");

ExampleMatcher matcher = ExampleMatcher.matching()
  .withIgnorePaths("lastname")
  .withIncludeNullValues()
  .withStringMatcherEnding();

Example<Person> example = Example.of(person, matcher);
```

50.5.11. Transakcyjność

- Przykład

```
public interface UserRepository extends CrudRepository<User, Long> {

  @Override
  @Transactional(timeout = 10)
  public List<User> findAll();

  // Further query method declarations
}



@Transactional(readOnly = true)
public interface UserRepository extends JpaRepository<User, Long> {

  List<User> findByLastname(String lastname);

  @Modifying
  @Transactional
  @Query("delete from User u where u.active = false")
  void deleteInactiveUsers();
}
```

50.5.12. Locking

```
interface UserRepository extends Repository<User, Long> {

  // Plain query method
  @Lock(LockModeType.READ)
  List<User> findByLastname(String lastname);
}
```

50.5.13. Audyt / Audit

- Przykład

```
class Customer {

  @CreatedBy
  private User user;

  @CreatedDate
  private DateTime createdDate;

  // … further properties omitted
}
```

50.5.14. AuditorAware

- Przykład

```
class SpringSecurityAuditorAware implements AuditorAware<User> {

  public User getCurrentAuditor() {

    Authentication authentication = SecurityContextHolder.getContext()
.getAuthentication();

    if (authentication == null || !authentication.isAuthenticated()) {
      return null;
    }

    return ((MyUserDetails) authentication.getPrincipal()).getUser();
  }
}
```

```
@Entity
@EntityListeners(AuditingEntityListener.class)
public class MyEntity {

}



@Configuration
@EnableJpaAuditing
class Config {

  @Bean
  public AuditorAware<AuditableUser> auditorProvider() {
    return new AuditorAwareImpl();
  }
}
```

50.6. Web support

50.6.1. Konfiguracja

- Przykład

```
@Configuration
@EnableWebMvc
@EnableSpringDataWebSupport
class WebConfiguration { }
```

DomainClassConverter

- Przykład

```
@Controller
@RequestMapping("/users")
public class UserController {

  @RequestMapping("/{id}")
  public String showUserForm(@PathVariable("id") User user, Model model) {

    model.addAttribute("user", user);
    return "userForm";
  }
}
```

HandlerMethodArgumentResolver dostęp do Pageable i Sort z poziomu parametrów requesta

- Przykład

```
@Controller
@RequestMapping("/users")
public class UserController {

  @Autowired UserRepository repository;

  @RequestMapping
  public String showUsers(Model model, Pageable pageable) {

    model.addAttribute("users", repository.findAll(pageable));
    return "users";
  }
}
```

Hypermedia wsparcie dla Pageables

- Przykład

```
@Controller
class PersonController {

  @Autowired PersonRepository repository;

  @RequestMapping(value = "/persons", method = RequestMethod.GET)
  HttpEntity<PagedResources<Person>> persons(Pageable pageable,
    PagedResourcesAssembler assembler) {

    Page<Person> persons = repository.findAll(pageable);
    return new ResponseEntity<>(assembler.toResources(persons), HttpStatus.OK);
  }
}
```

Querydsl web support / QuerydslPredicateArgumentResolver.

- Przykład

```
?firstname=Dave&lastname=Matthews

=>

QUser.user.firstname.eq("Dave").and(QUser.user.lastname.eq("Matthews"))
```

- Przykład

```
@Controller
class UserController {

  @Autowired UserRepository repository;

  @RequestMapping(value = "/", method = RequestMethod.GET)
  String index(Model model, @QuerydslPredicate(root = User.class) Predicate predicate,
          Pageable pageable, @RequestParam MultiValueMap<String, String> parameters) {

    model.addAttribute("users", repository.findAll(predicate, pageable));

    return "index";
  }
}



interface UserRepository extends CrudRepository<User, String>,
                                 QueryDslPredicateExecutor<User>,
                                 QuerydslBinderCustomizer<QUser> {

  @Override
  default public void customize(QuerydslBindings bindings, QUser user) {

    bindings.bind(user.username).first((path, value) -> path.contains(value))
    bindings.bind(String.class)
      .first((StringPath path, String value) -> path.containsIgnoreCase(value));
    bindings.excluding(user.password);
  }
}
```

51. NoSql

52. MongoDB

Zależność :

```
spring-boot-starter-data-mongodb
```

52.1. Przykładowa encja

```
@Document()
public class Person {
@Id
private String id;
private String name;
@DBRef
private List<Invoice> invoices;
```

52.2. Parametryzacja

```
spring.data.mongodb.host= # the db host
spring.data.mongodb.port=27017 # the connection port
spring.data.mongodb.uri=mongodb://localhost/test # connection URL
spring.data.mongo.repositories.enabled=true # if SD enabled
```

52.3. MongoTemplate

```
@Component
public class MyBean {

    private final MongoTemplate mongoTemplate;

    @Autowired
    public MyBean(MongoTemplate mongoTemplate) {
        this.mongoTemplate = mongoTemplate;
    }

    // ...

}
```

52.4. Konfiguracja

```
@EnableMongoRepositories(basePackages = "com.acme.repositories.mongo")
interface Configuration { }

@Configuration
public class AppConfig {

  /*
   * Use the standard Mongo driver API to create a com.mongodb.Mongo instance.
   */
   public @Bean Mongo mongo() throws UnknownHostException {
       return new Mongo("localhost");
   }
}
@Configuration
public class AppConfig {

    /*
     * Factory bean that creates the com.mongodb.Mongo instance
     */
     public @Bean MongoClientFactoryBean mongo() {
          MongoClientFactoryBean mongo = new MongoClientFactoryBean();
          mongo.setHost("localhost");
          return mongo;
     }
}



@Configuration
public class MongoConfiguration {

  public @Bean MongoDbFactory mongoDbFactory() throws Exception {
    UserCredentials userCredentials = new UserCredentials("joe", "secret");
    return new SimpleMongoDbFactory(new Mongo(), "database", userCredentials);
  }

  public @Bean MongoTemplate mongoTemplate() throws Exception {
    return new MongoTemplate(mongoDbFactory());
  }
}
```

52.5. Spring Data MongoDB repozytoria

```
public interface CityRepository extends Repository<City, Long> {

    Page<City> findAll(Pageable pageable);

    City findByNameAndCountryAllIgnoringCase(String name, String country);

}
```

52.6. Embedded Mongo

Zależność maven:

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>
        <dependency>
            <groupId>de.flapdoodle.embed</groupId>
            <artifactId>de.flapdoodle.embed.mongo</artifactId>
            <version>1.48.0</version>
            <scope>test</scope>
        </dependency>
```

Zależność gradle :

```
    compile 'org.springframework.boot:spring-boot-starter-data-mongodb'
    compile group: 'de.flapdoodle.embed', name: 'de.flapdoodle.embed.mongo', version:
'1.50.5'
```

Health status :

```
{

    "status": "UP",
    "diskSpace": {
        "status": "UP",
        "total": 219353915392,
        "free": 121737756672,
        "threshold": 10485760
    },
    "mongo": {
        "status": "UP",
        "version": "2.6.10"
    }

}
```

53. Neo4j

53.1. Połączenie z bazą Neo4j

Neo4jSession, Session, Neo4jOperations

53.2. Neo4jTemplate

```
@Component
public class MyBean {

    private final Neo4jTemplate neo4jTemplate;

    @Autowired
    public MyBean(Neo4jTemplate neo4jTemplate) {
        this.neo4jTemplate = neo4jTemplate;
    }

    // ...

}
```

```
spring.data.neo4j.uri=http://my-server:7474
spring.data.neo4j.username=neo4j
spring.data.neo4j.password=secret
```

53.3. Embedded mode

Zależność :

```
org.neo4j:neo4j-ogm-embedded-driver
```

Ustawienia :

```
spring.data.neo4j.uri=file://var/tmp/graph.db
```

53.4. Spring Data Neo4j repozytoria

@EnableNeo4jRepositories(basePackages = "com.example.myapp.repository")
@EnableTransactionManagement

```
public interface CityRepository extends GraphRepository<City> {

    Page<City> findAll(Pageable pageable);

    City findByNameAndCountry(String name, String country);

}
```

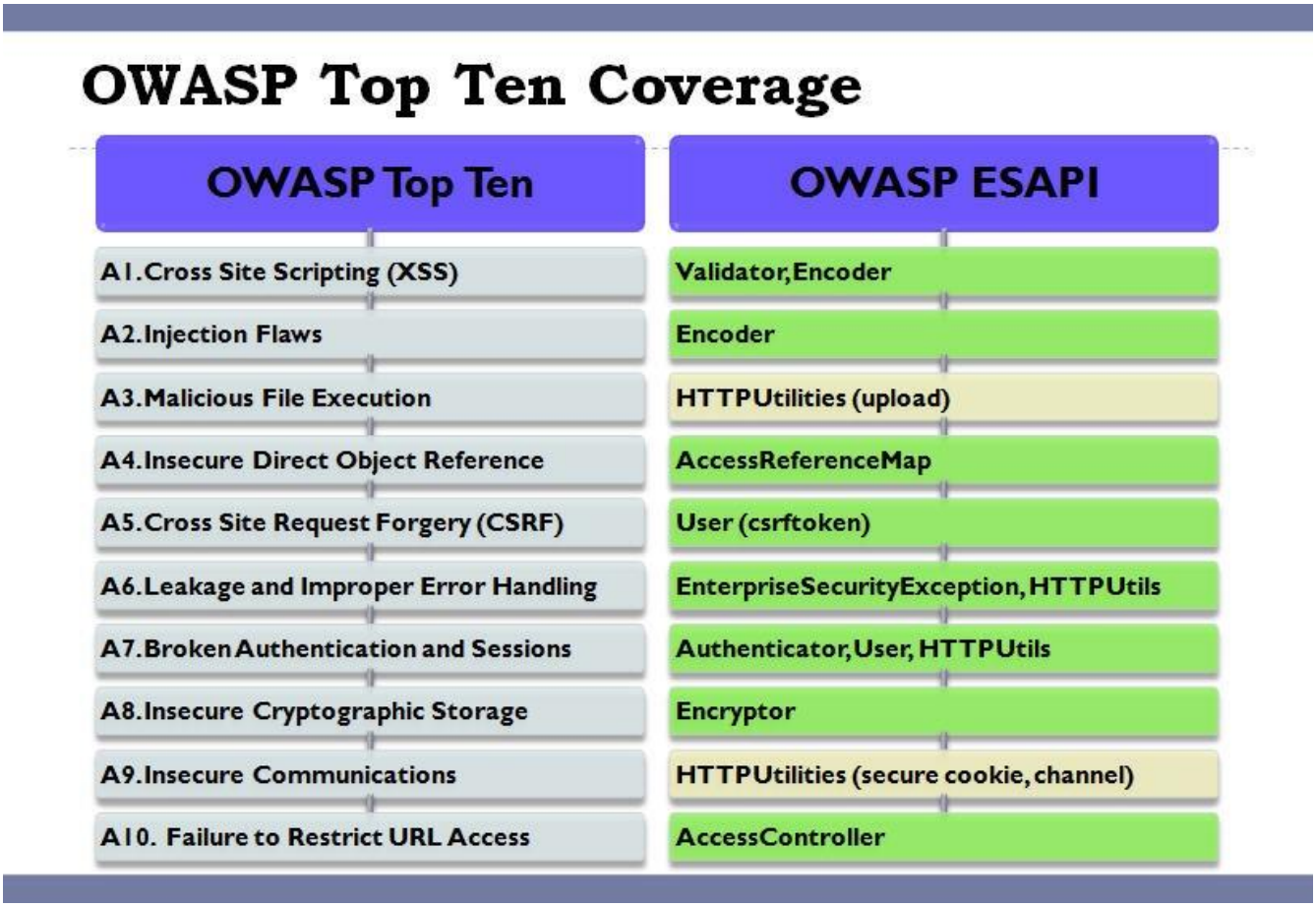
54. Using ResourceBundles for Internationalization (I18N)

55. Spring Security

- **wszechstronność metod autoryzacji i autentykacji**
- **zabezpieczenie przez atakami (OWASP top ten)**
- **integracja ze Spring MVC i Servlet API 3 i 3.1**
- **od wersji 3.2 wsparcie dla JavaConfig**
- **zabezpieczenie przed CRFS, session fixation, clickjacking, nagłówki security**
- **automatyczny resolver dla pobierania Principle z żądania**
- **integracje z usługami asynchronicznymi w Spring MVC**
- **przenośność rozwiązania**
- **dojrzałość**
- **pewność**

- **integracja z różnymi protokołami i rozwiązaniami jak :**
  - openID
  - OAuth
  - LDAP
  - SSO (CAS)
  - RMI , HttpInvoker
  - JAAS
- Kerberos
  - HTTP X.509
  - i wiele innych

55.1. OWASP Top Ten

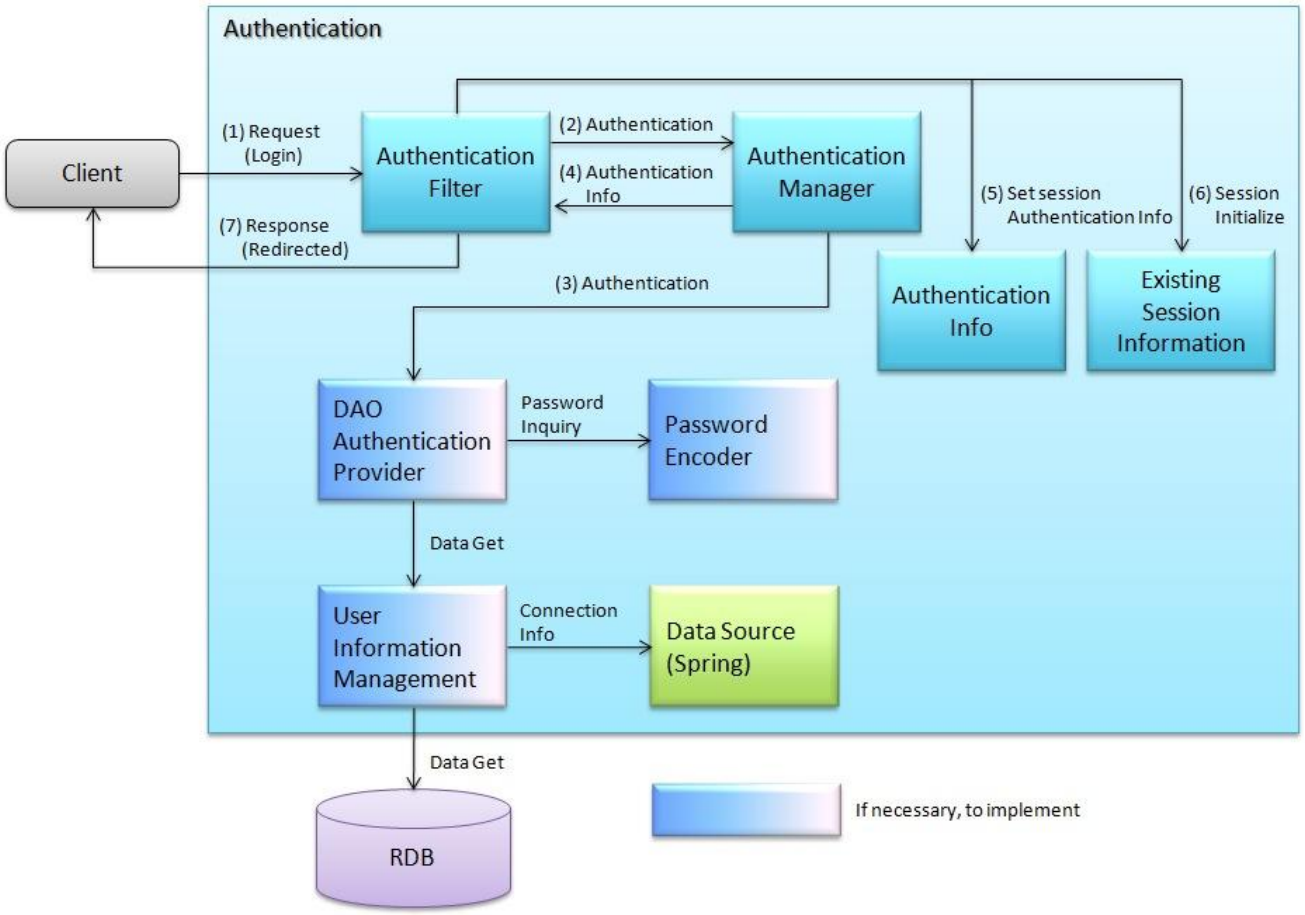


55.2. Authentication : autoryzacja: uwierzytelnianie

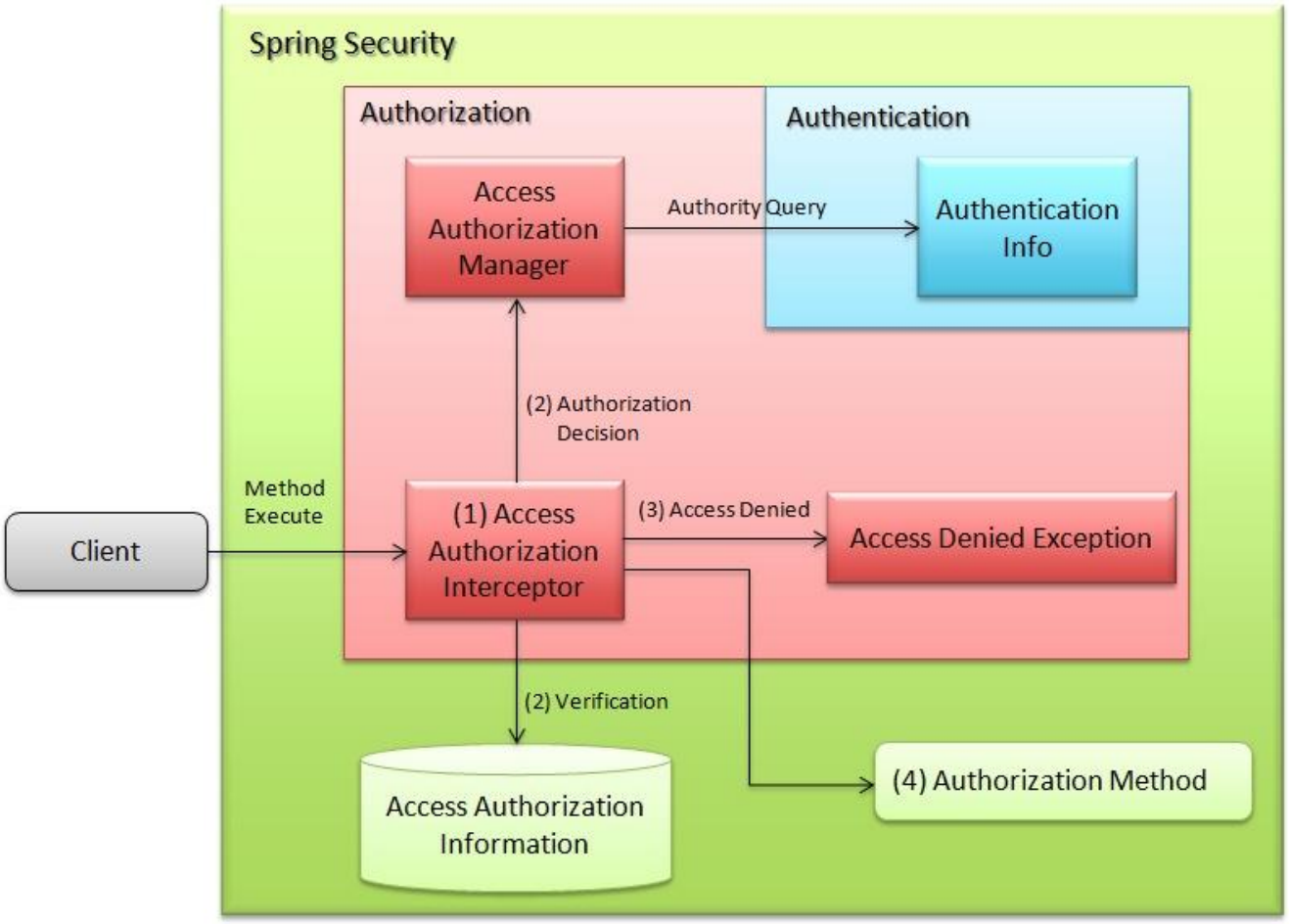
Zapewnia kontrolowany dostęp do zasobów , bazuje na identyfikacji i uwierzytelnianiu.

Autoryzacja to proces przyznawania uprawnień uwierzytelnionemu użytkownikowi. Dzięki temu użytkownik dostaje dostęp do określonych zasobów aplikacji czy systemu. Proces ten następuje co procesie uwierzytelnienia. Uprawnienia zwykle są w postaci ról.

source: http://terasolunaorg.github.io/guideline/5.0.0.RELEASE/en/Security/Authentication.html



source:http://terasolunaorg.github.io/guideline/5.0.0.RELEASE/en/Security/Authorization.html



55.3. Access control: Kontrola dostępu

Kontrola dostępu polega na kontrolowaniu dostępu do określonego zasobu. Wymaga to ustalenia czy użytkownikowi należy przyznać dostęp do określonego zasobu. Proces ustalania kontroli dostępu związany jest z porównaniem atrybutu dostępu do danego zasobu z uprawnieniami lub innymi cechami użytkownika.

- Autoryzacja **lub** kontrola dostępu** - procesy i środki dające dostęp do zasobów do których użytkownik jest zobligowany. Umożliwia na wykonywanie poszczególnych operacji lub dostęp do danych dla użytkowników, którzy faktycznie mają do nich dostępem

55.4. Standartowe podejście

Zależności :

55.5. Maven:

```
<dependencies>
<!-- ... other dependency elements ... -->
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-web</artifactId>
<version>3.2.4.RELEASE</version>
</dependency>
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-config</artifactId>
<version>3.2.4.RELEASE</version>
</dependency>
</dependencies>
```

55.6. Gradle :

```
dependencies {
compile 'org.springframework.security:spring-security-web:4.0.3.RELEASE'
compile 'org.springframework.security:spring-security-config:4.0.3.RELEASE'
}
```

55.7. Moduły

- **Core spring-security-core** - auth i access-control, remoting support - ogólnie jak to 'core' jest niezbędny do działania czegokolwiek w spring security framework. Zapewnia zabezpieczenie dla aplikacji standalone, zdalnych klientów, i metod biznesowych, JDBC oraz obsługę kryptografii (Crypto)
- **WEB spring-security-web** - zawiera filtry, które możemy modelować w łańcuch (Chain of responsiblity pattern) oraz integracje z Servlet API i URL-based access-control
- **Config spring-security-config** - zawiera Xml namespace dla wygodnej konfiguracji z poziomu XML znanej ze Springa oraz JavaConfig support
- **LDAP - spring-security-ldap** - autoryzacja oparta na protokole LDAP
- **ACL - spring-security-acl** - autoryzacja oparta na poziomie dostępu do obiektów domenowych. Znamy takie podejście z systemów Linux czyli kto i w jakim stopniu ma dostęp do zasobu.
- **CAS - spring-security-cas** - integracje z systemami SSO.
- **OpenID - spring-security-openid** - wsparcie dla protokołu openID. Autoryzacja użytkownika po przez zewnętrzne serwery OpenID np Google

55.8. Sposoby logowania

- Login Form (login/password) authentication
- Basic, Digest authentication (HTTP basic authentication)

- Rememeber Me (cookie)
- Ldap
- Kerberos , Active Directory
- JAAS
- X.509
- CAS
- OpenId
- Oauth2

55.9. Zabezpieczanie dostępu do adresów URL

```
@Override
protected void configure(HttpSecurity http) throws Exception {
http.authorizeRequests()
.antMatchers("/user/register").permitAll()
.antMatchers("/user/autologin").access("hasRole('ROLE_USER')")
.antMatchers("/user/delete").access("hasRole('ROLE_ADMIN')")
.antMatchers("/img/**").permitAll()
.antMatchers("/images/**").permitAll()
.anyRequest().authenticated()
.and()
.formLogin().loginPage("/login").failureUrl("/login?error").permitAll()
.and()
.logout().logoutUrl("/logout")
.logoutSuccessHandler(logoutHandler)
.deleteCookies("JSESSIONID")
.and()
.rememberMe().key(applicationSecret)
.tokenValiditySeconds(31536000);
}
```

55.10. Zabezpiecznie ścieżek dostępu

- access(String) – jeśli wartością SpEL jest true - dostęp
- anonymous() - umożliwia dostęp anonimowych użytkownikom
- authenticated() - umożliwia dostęp uwierzytelnionym użytkownikom
- denyAll() - bezwarunkowa odmowa dostępu
- fullyAuthenicated() - dostęp jedynie dla pełno uwierzytelnionych użytkowników (niezapamiętanym)
- hasAnyAuthority(String ...) - dostęp jeśli jedno z podanych uprawnień jest przypisane do użytkownika
- hasAnyRole(String ...) - umożliwia dostęp jeśli user posiada jedno z wymienionych ról

- hasAuthority(String) – umożliwia dostęp jeśli user posiada wybrane uprawnienie
- hasIpAddress(String) – jeśli user na zadany adres ip - dostęp
- hasRole(String) – umożliwia dostęp jeśli user ma zadaną role
- not() - neguje efekt wszystkich powyższych metod
- permitAll() - bezwarunkowy dostęp
- rememberMe() - umożliwia dostęp dla użytkowników z opcją remember-me

55.11. Zabezpieczanie za pomocą SpEL : zwiększamy elastyczność

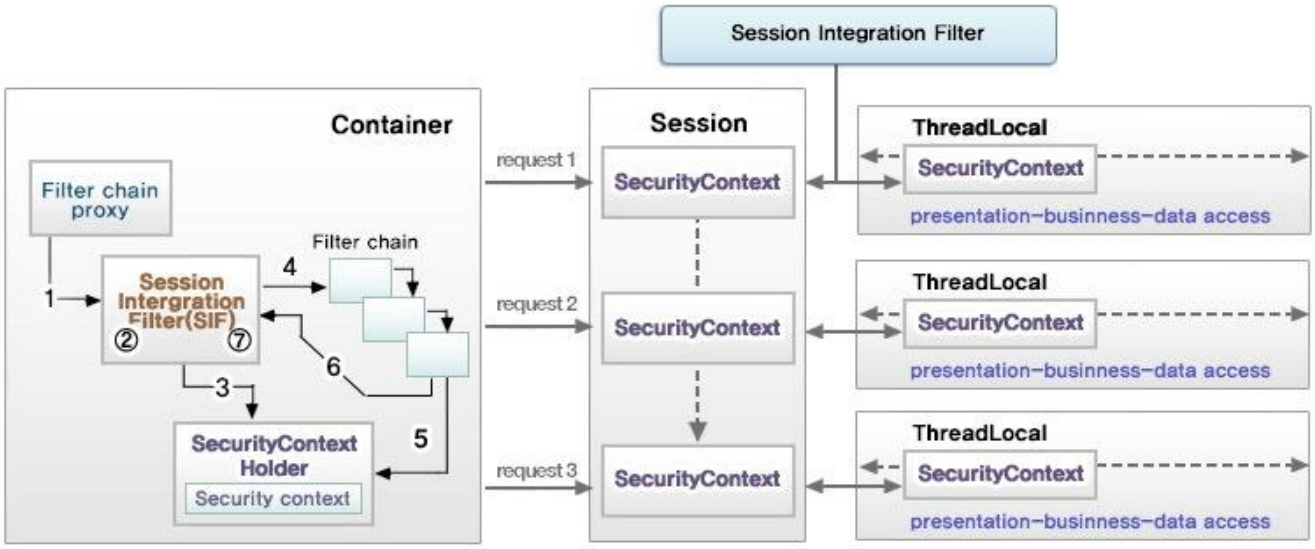
- authentication – obiekt authentication użytkownika
- denyAll – zawsze false
- hasRole(String role) – true, jeśli użytkownik posiada taką rolę
- hasAnyRole(lista ról) – true, jeśli user posiada którąś z ról
- hasIpAddress(adres ip) – jeśli user jest przydzielony do tego adresu wtedy true
- isAnonymous() - true, jeśli user jest anonimowy
- isAuthenticated() - true, jeśli user nie jest anonimowy
- isFullyAuthenticated() - true , jeśli użytkownik jest w pełni zalogowany (!rememberMe)
- isRememberMe() - true, jeśli user jest zalogowany z opcją remember-me
- permitAll – zawsze true
- principal – obiekt principal danego użytkownika
    - Przykład

```
antMatchers("/author/create").access("hasRole('ROLE_ADMIN') or hasRole('ROLE_DBA')")
antMatchers("/messageDelete*").access("hasRole('ROLE_ADMIN) or
hasIpAddress('127.0.0.1) or hasIpAddress('0:0:0:0:0:0:0:1')")
```

56. Koncepcja

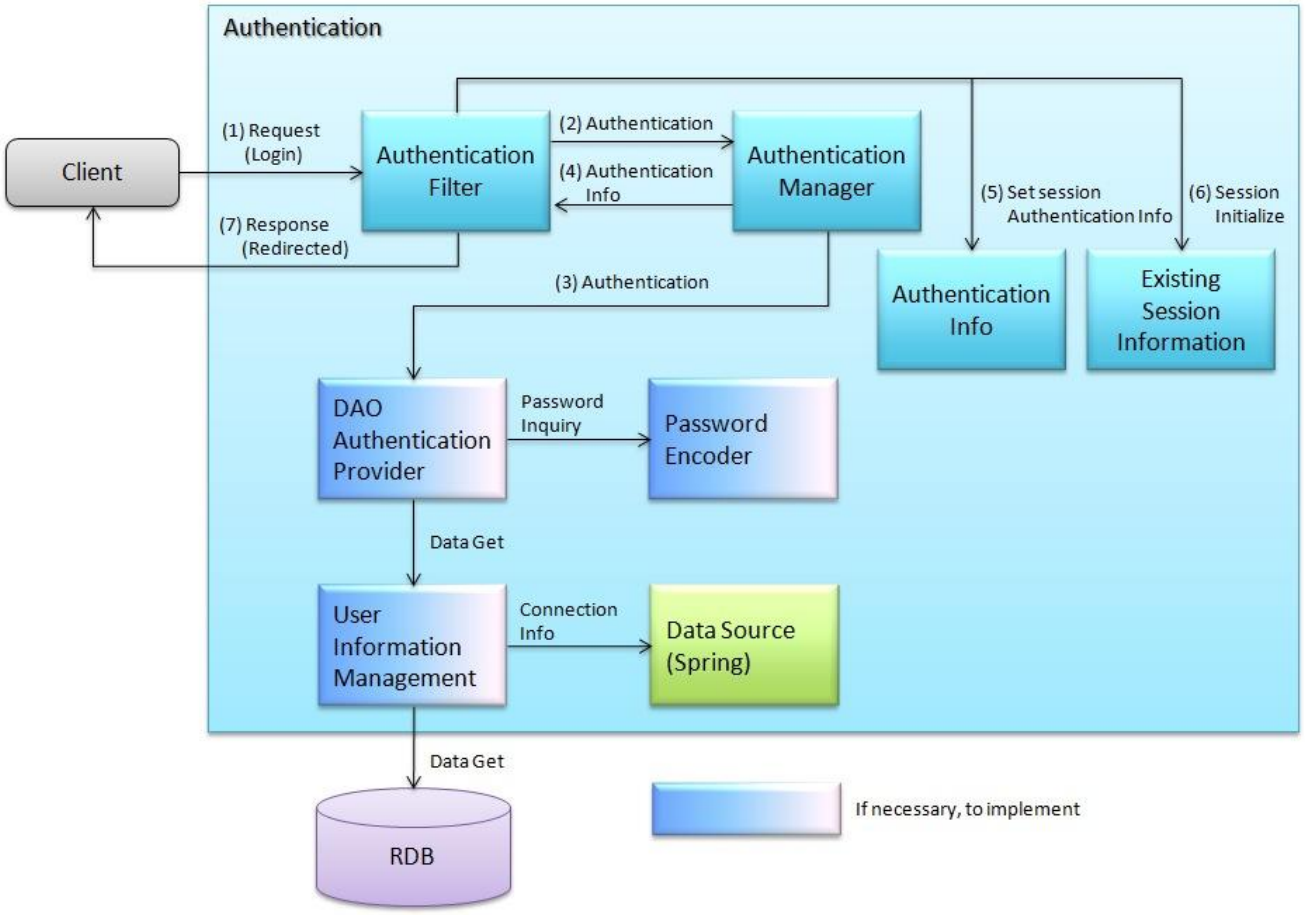
- SecurityContextHolder → SecurityContext – dostęp wielowątkowy
- SecurityContext → Authentication

source:http://dev.anyframejava.org/



- Authentication → principal
- Principal – główna koncepcja użytkownika dla uwierzytelniania
- UserDetails - user zrozumiały dla SpringSecurity
- User – zwykła klasa domenowa

source: http://terasolunaorg.github.io/guideline/5.0.0.RELEASE/en/Security/Authentication.html



- Przykład

```
private User getLoggedUser(){
String login = ((UserDetails (SecurityContextHolder.getContext()).getAuthentication()
.getPrincipal()).getUsername();
return userDAO.findByLogin(login);
}
```

56.1. UserDetailService (read-only)

```
public interface UserDetailsService {
UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

- Dostarcza usługę uwierzytelnienia dla AuthenticationManager

```
@Service("userAccountDetailsService")
public class UserAccountDetailsService implements UserDetailsService {
  @Override
  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
  UserAccount account = getUserAccount(username);
  return new User(account.getUsername(),account.getPassword(),AuthorityUtils
.createAuthorityList( account.getAuthority()));
}

public UserAccount getUserAccount(String username) {
try {
    TypedQuery<UserAccount> query = UserAccount.findUserAccountsByUsernameEquals
(username);
    return query.getSingleResult();
     } catch (EmptyResultDataAccessException ex) {
              throw new UsernameNotFoundException("Could not find user " + username,
ex);
}
}
}
```

56.2. UserDetails

- **accountExpired(boolean)** – określa, czy konto wygasło czy nie
- **accountLocked(boolean)** – określa, czy konto zostało zablokowane czy też nie
- **and()** - łączenie w łańcuch
- **authorities(GrantedAuthority ...)** - określa jedno lub więcej uprawnień nadanych użytkownikowi
- **credentialsExpired(boolean)** - określa, czy dane wygasły czy też nie

- **disabled(boolean)** – określa, czy konto jest włączone czy też nie
- **password(String)** - hasło użytkownika
- **roles(String ...)** - jedna lub więcej ról przypisanych do użytkownika

56.3. @EnableWebSecurity

- **@EnableWebSecurity** - włączamy ustawienia bezpieczeństwa wyłącznie dla aplikacji typu web.

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

- **@EnableWebSecurity** żyje w symbiozie z WebSecurityConfigurer.
- **WebSecurityConfigurerAdaper** rozszerza WebSecurityConfigurer dla potrzeb dewelopera.
- **configure(WebSecurity)** - jej nadpisanie umożliwia konfigurację łańcucha filtrów
- **configure(HttpSecurity)** - jej nadpisanie umożliwia konfigurację sposobu zabezpieczenia żądań za pomocą interceptorów
- **configure(AuthenticationManagerBuilder)** – jej nadpisanie umożliwia konfigurację usług szczegółów użytkownika.
  - Przykład min

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
auth.inMemoryAuthentication() .withUser("user").password("password").roles("USER");
}
```

```
protected void configure(HttpSecurity http) throws Exception {
http.authorizeRequests().anyRequest().authenticated().and()
.formLogin()
.and()
.httpBasic();
}
```

56.4. Login Form

- XML : form-login:

```
loginProcessingUrl: /j_spring_security_check
usernameParameter: j_username
passwordParameter: j_password
```

- Java Configuration :

```
formLogin():
loginProcessingUrl: /login
usernameParameter: username
passwordParameter: password
```

```
protected void configuare(HttpSecurity http) throws Exception {
http.authorizeRequests().anyRequest().authenticated().and().formLogin().
loginPage("/login") .permitAll();
}
```

- View

```
<c:url value="/login" var="loginUrl"/>
<form action="${loginUrl}" method="post">
<c:if test="${param.error != null}">
<p>Invalid username and password </p>
</c:if>
<c:if test="${param.logout != null}">
<p> You have been logged out. </p>
</c:if>
<p>
<label for="username">Username</label>
<input type="text" id="username" name="username"/>
</p>
<p>
<label for="password">Password</label>
<input type="password" id="password" name="password"/>
</p>
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
<button type="submit" class="btn">Log in</button>
</form>
```

56.5. Zabezpieczanie widoków widoków : Thymeleaf

- **sec:authentication** - generuje właściwości obiektu uwierzytelnienia.
- **sec:authorize** - warunkowo generuje zawartość w oparciu o wartość wyrażenia.
- **sec:authorize-acl** - warunkowo generuje zawartość w oparciu o wartość wyrażenia
- **sec:authorize-expr** - alias dla atrybutu sec:authorize

- **sec:authorize-url** - warunkowo generuje zawartość w oparciu o reguły zabezpieczeń powiązane z daną ścieżką URL.

```
<div sec:authorize="hasRole('ROLE_ADMIN')">
This content is only shown to administrators.
</div>

<div sec:authorize="hasRole('ROLE_USER')">
This content is only shown to users.
</div>

Logged user: <span sec:authentication="name">Bob</span>
Roles: <span sec:authentication="principal.authorities">[ROLE_USER, ROLE_ADMIN]</span>
```

```
<div th:with= "currentUser=${#httpServletRequest.userPrincipal?.name}">
<div th:if= "${currentUser != null}">
<form th:action="@{/logout}" method="post”>
<input type= "submit" value="Log out" />
</form>
<p th:text="${currentUser}”>sample_user</p>
</div>
```

56.6. Backend

```
public class BackendAuth {

@Autowired
private AuthenticationManager authenticationManager;

public boolean login(String login, String password) {
 Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(login, password));
  boolean isAuthenticated = isAuthenticated(authentication);
  if (isAuthenticated) {
   SecurityContextHolder.getContext().setAuthentication(authentication);
   }
return isAuthenticated;
}
public boolean isAuthenticated(Authentication authentication) {
  return authentication != null && !(authentication instanceof
AnonymousAuthenticationToken) && authentication.isAuthenticated();
}

}
```

56.7. @EnableGlobalMethodSecurity

Konsola:

```
The default AuthenticationManager has a single user (‘user’ username and random
password, printed at INFO level when the application starts up)

Using default security password: 78fa095d-3f4c-48b1-ad50-e24c31d5cf35
```

57. Caching

57.1. Możliwi dostawcy

- **Generic
- **JCache (JSR-107)**
- **EhCache 2.x**
- **Hazelcast**
- **Infinispan**
- **Couchbase**
- **Redis**
- **Caffeine**
- **Guava**
- **Simple**

57.2. CacheManager

```
@Bean
public CacheManagerCustomizer<ConcurrentMapCacheManager> cacheManagerCustomizer() {
    return new CacheManagerCustomizer<ConcurrentMapCacheManager>() {
        @Override
        public void customize(ConcurrentMapCacheManager cacheManager) {
            cacheManager.setCacheNames(Arrays.asList("one", "two"));
        }
    };
}
```

57.3. EhCache 2.x

spring.cache.ehcache.config=classpath:config/another-config.xml

```
@Bean
public EhCacheManagerFactoryBean getEhCacheFactory() {
    EhCacheManagerFactoryBean factoryBean = new EhCacheManagerFactoryBean();
    factoryBean.setConfigLocation(new ClassPathResource("ehcache.xml"));
    factoryBean.setShared(true);
    return factoryBean;
}

@Bean
public CacheManager getEhCacheManager() {
    return new EhCacheCacheManager(getEhCacheFactory().getObject());
}
```

Przykład : ehcache.xml

```
<ehcache xsi:noNamespaceSchemaLocation="ehcache.xsd"
updateCheck="true" monitoring="autodetect" dynamicConfig="true"
maxBytesLocalHeap="150M">

<diskStore path="java.io.tmpdir"/>

<cache name="person" maxBytesLocalHeap="100M"
eternal="false"
timeToIdleSeconds="300"
overflowToDisk="true"
maxElementsOnDisk="1000"
memoryStoreEvictionPolicy="LRU"/>
```

Zależności :

```
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
</dependency>

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

<dependency>
    <groupId>org.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>2.8.+</version>
</dependency>
```

```
@Bean
public JCacheManagerCustomizer cacheManagerCustomizer() {
    return cm -> {
        MutableConfiguration<Object, Object> configuration = new
MutableConfiguration<>()
                .setExpiryPolicyFactory(CreatedExpiryPolicy
                .factoryOf(Duration.ONE_HOUR))
                .setStoreByValue(false)
                .setStatisticsEnabled(true);
        cm.createCache("person", configuration);
        cm.createCache("boot-versions", configuration);
    };
}
```

58. Caffeine / Guava

Zależność:

```
compile 'org.springframework.boot:spring-boot-starter-cache'
compile 'com.github.ben-manes.caffeine:caffeine'
```

Ustawienia:

@EnableCaching

```
spring.cache.caffeine.spec=recordStats,maximumSize=500,expireAfterAccess=600s
```

59. Messaging

60. JMS

60.1. Rodzaje :

- **Point-to-point messaging:**

Wiadomość dostaje tylko jeden odbiorca

- **Publish-Subscribe messaging:**

Kopie wiadomość dostaje wiele konsumentów

60.2. Typy konsumentów :

- **Message listeners**

Model asynchroniczny

- **receive**

Model synchroniczny

60.3. JmsTemplate

61. ActiveMQ support

Zależność

```
compile('org.springframework.boot:spring-boot-starter-activemq')
```

Dodajemy możliwość korzystania z puli:

```
 compile group: 'org.apache.activemq', name: 'activemq-pool', version: '5.14.0'
```

ActiveMQAutoConfiguration.class

Ustawienia :

```
spring.activemq.*
```

Przyklad :

```
spring.activemq.in-memory=false
spring.activemq.pool.enabled=true
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.pool.configuration.max-connections=10
spring.activemq.password=admin
spring.activemq.user=admin
spring.jms.listener.concurrency=10
management.health.jms.enabled=true
```

Health status:

```
{
  "status": "UP",
  "jms": {
    "status": "UP",
    "provider": "ActiveMQ"
  },
  "diskSpace": {
    "status": "UP",
    "total": 219353915392,
    "free": 121812205568,
    "threshold": 10485760
  }
}
```

- Programowe tworzenie brokera

```
@Configuration
public class ActiveMqConfiguration {

    public static final String ADDRESS = "vm://localhost";

    private BrokerService broker;

    @Bean(name = "myQueueDestination")
    public Destination bookMgrQueueDestination(@Value("${jms.myqueue.name}") String
myQueueName) throws JMSException {
        return new ActiveMQQueue(myQueueName);
    }

    @PostConstruct
    public void startActiveMQ() throws Exception {
        broker = new BrokerService();
        broker.setBrokerName("activemq-broker");
        broker.setDataDirectory("target");
        broker.addConnector(ADDRESS);
        broker.setUseJmx(false);
        broker.setUseShutdownHook(false);
        broker.start();
    }

    @PreDestroy
    public void stopActiveMQ() throws Exception {
        broker.stop();
    }

    @Bean
    public ConnectionFactory connectionFactory() {
        return new ActiveMQConnectionFactory(ADDRESS + "?broker.persistent=false");
    }
}
```

- Programowe tworzenie listenera i JmsTemplate

```
@EnableJms
@Configuration
public class JmsConfiguration {

    @Bean
    JmsTemplate jmsTemplate(ConnectionFactory connectionFactory) {
    JmsTemplate jmsTemplate = new JmsTemplate(connectionFactory);
    jmsTemplate.setSessionTransacted(true);
    return jmsTemplate ;
    }

    @Bean
    public ConnectionFactory connectionFactory() {
        return new CachingConnectionFactory(new
ActiveMQConnectionFactory("tcp://localhost:61616"));
    }

    @Bean
    public PlatformTransactionManager transactionManager() throws NamingException {
        return new JmsTransactionManager(connectionFactory());
}

    @Bean(name = "DefaultJmsListenerContainerFactory")
    public DefaultJmsListenerContainerFactory
provideJmsListenerContainerFactory(PlatformTransactionManager transactionManager) {
        DefaultJmsListenerContainerFactory factory = new
DefaultJmsListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory());
        factory.setTransactionManager(transactionManager);
        factory.setConcurrency("5-10");
        factory.setSessionAcknowledgeMode(Session.CLIENT_ACKNOWLEDGE);
        factory.setSessionTransacted(true);
        return factory;
}

}
```

- Tworzenie odbiorcy

```
@Component
public class Consumer {
    @JmsListener(destination = "sample.queue")
    public void receiveQueue(String text) {
        System.out.println(text);
    }

}
```

- Tworzenie producenta

```
@Component
public class Producer implements CommandLineRunner {

    @Autowired
    private JmsMessagingTemplate jmsMessagingTemplate;

    @Autowired
    private Queue queue;

    @Override
    public void run(String... args) throws Exception {
        send("Sample message");
        System.out.println("Message was sent to the Queue");
    }

    public void send(String msg) {
        this.jmsMessagingTemplate.convertAndSend(this.queue, msg);
    }

}
```

Przykład 2:

```
public class JmsQueueSender {

    private JmsTemplate jmsTemplate;
    private Queue queue;

    public void setConnectionFactory(ConnectionFactory cf) {
        this.jmsTemplate = new JmsTemplate(cf);
    }

    public void setQueue(Queue queue) {
        this.queue = queue;
    }

    public void simpleSend() {
        this.jmsTemplate.send(this.queue, new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage("hello queue world");
            }
        });
    }
}
```

- Tworzenie kolejki

```
@Bean
public Queue queue() {
    return new ActiveMQQueue("sample.queue");
}
```

- Tworzenie fabryki (czysty Spring)

```
<bean id="connectionFactory"
class="org.apache.activemq.spring.ActiveMQConnectionFactory"
p:brokerURL="tcp://localhost:61616"/>
```

i ponawianie jeśli to konieczne :

```
<property name="redeliveryPolicy">
<bean class="org.apache.activemq.RedeliveryPolicy">
<property name="maximumRedeliveries" value="3"/>
</bean>
</property>
```

NOTE : redelivery_programming_way

lub za pomocą ActiveMq namespace

```
<?xml version="1.0" encoding="UTF-8"?>
  <beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:amq="http://activemq.apache.org/schema/core"
    xsi:schemaLocation="http://activemq.apache.org/schema/core
      http://activemq.apache.org/schema/core/activemq-core.xsd
      http://www.springframework.org/schema/jms
      http://www.springframework.org/schema/jms/spring-jms.xsd
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

  <amq:connectionFactory id="connectionFactory"  brokerURL="tcp://localhost:61616"/>

</beans>
```

- Deklaracja kolejki

```
<bean id="queue" class="org.apache.activemq.command.ActiveMQQueue" c:_="my.queue" />
```

lub namespace:

```
<amq:queue id="myQueue" physicalName="my.queue" />
```

- Deklaracja tematu

```
<bean id="topic" class="org.apache.activemq.command.ActiveMQTopic" c:_="myTopic.queue"
/>
```

lub namespace:

```
<amq:topic id="myTopic" physicalName="my.topic" />
```

- JmsTemplate

```
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate" c:_-
ref="connectionFactory" />
```

```
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
<property name="connectionFactory"  ref="connectionFactory" />
<property name="defaultDestination" ref="destination" />
</bean>
```

- Wysyłanie wiadomości (send())

```
jmsTemplate.send(
   new MessageCreator() {
   ...
   }
);
```

- Wysyłanie wiadomości (convertAndSend())
  - **MessageConverter**
  - **MappingJacksonMessageConverter (JSON)**

```
<bean id="messageConverter"
class="org.springframework.jms.support.converter.MappingJacksonMessageConverter" />

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"
      c:_-ref="connectionFactory"
      p:defaultDestinationName="my.queue"
      p:messageConverter-ref="messageConverter" />
```

- **MappingJackson2MessageConverter (JSON)**
- **MarshallingMessageConverter (JAXB)**
- **SimpleMessageConverter (String→TextMessage)**

Przykład :

```
public void sendWithConversion() {
    Map map = new HashMap();
    map.put("Name", "Mark");
    map.put("Age", new Integer(47));
    jmsTemplate.convertAndSend("testQueue", map, new MessagePostProcessor() {
        public Message postProcessMessage(Message message) throws JMSException {
            message.setIntProperty("AccountID", 1234);
            message.setJMSCorrelationID("123-00001");
            return message;
        }
    });
}
```

- Odbieranie wiadomości

```
public Alert receiveMyAlert() {
  try {
    ObjectMessage receivedMessage = (ObjectMessage) jmsOperations.receive();
    return (Alert) receivedMessage.getObject();
  } catch (JMSException jmsException) {
      throw JmsUtils.convertJmsAccessException(jmsException);
  }
}
```

61.1. Sterowanie komunikatami @MessageDriven

```
@MessageDriven(mappedName="jms/my.alert.queue")
  public class AlertHandler implements MessageListener {

  @Resource
  private MessageDrivenContext mdc;

  public void onMessage(Message message) {
    ...
  }

  public void handleAlert(Alert alert){
  }
}
```

MessageListener nie jest konieczny

```
<jms:listener-container connection-factory="connectionFactory">
  <jms:listener destination="my.alert.queue"
        ref="spittleHandler" method="handleAlert" />
</jms:listener-container>
```

Zależność:

```
spring-boot-starter-activemq
```

```
compile("org.springframework.boot:spring-boot-starter")
compile("org.springframework:spring-jms")
compile("org.apache.activemq:activemq-broker")
```

```
spring.activemq.broker-url=tcp://192.168.1.210:9876
spring.activemq.user=admin
spring.activemq.password=secret
```

61.2. @EnableJms

61.3. Wysyłanie wiadomości

```
@Component
public class MyBean {

    private final JmsTemplate jmsTemplate;

    @Autowired
    public MyBean(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    // ...

}
```

61.4. Odbieranie wiadomości

```
@Component
public class MyBean {

    @JmsListener(destination = "someQueue")
    public void processMessage(String content) {
        // ...
    }

}
```

61.5. Własny JmsListenerContainer

```
@Configuration
static class JmsConfiguration {

    @Bean
    public DefaultJmsListenerContainerFactory myFactory(
            DefaultJmsListenerContainerFactoryConfigurer configurer) {
        DefaultJmsListenerContainerFactory factory =
                new DefaultJmsListenerContainerFactory();
        configurer.configure(factory, connectionFactory());
        factory.setMessageConverter(myMessageConverter());
        return factory;
    }

}


@Component
public class MyBean {

    @JmsListener(destination = "someQueue", containerFactory="myFactory")
    public void processMessage(String content) {
        // ...
    }

}
```

61.6. Monitoring

- JMX
- Hawtio

Zależności :

```
compile group: 'io.hawt', name: 'hawtio-springboot', version: '1.4.65'
compile group: 'io.hawt', name: 'hawtio-core', version: '1.4.65'
compile group: 'org.jolokia', name: 'jolokia-core', version: '1.3.4'
```

Przykładowa konfiguracja:

```
@Configuration
@EnableHawtio
public class HawtioConfiguration  {
    static {
        System.setProperty(AuthenticationFilter.HAWTIO_AUTHENTICATION_ENABLED,
"false");
}
}
```

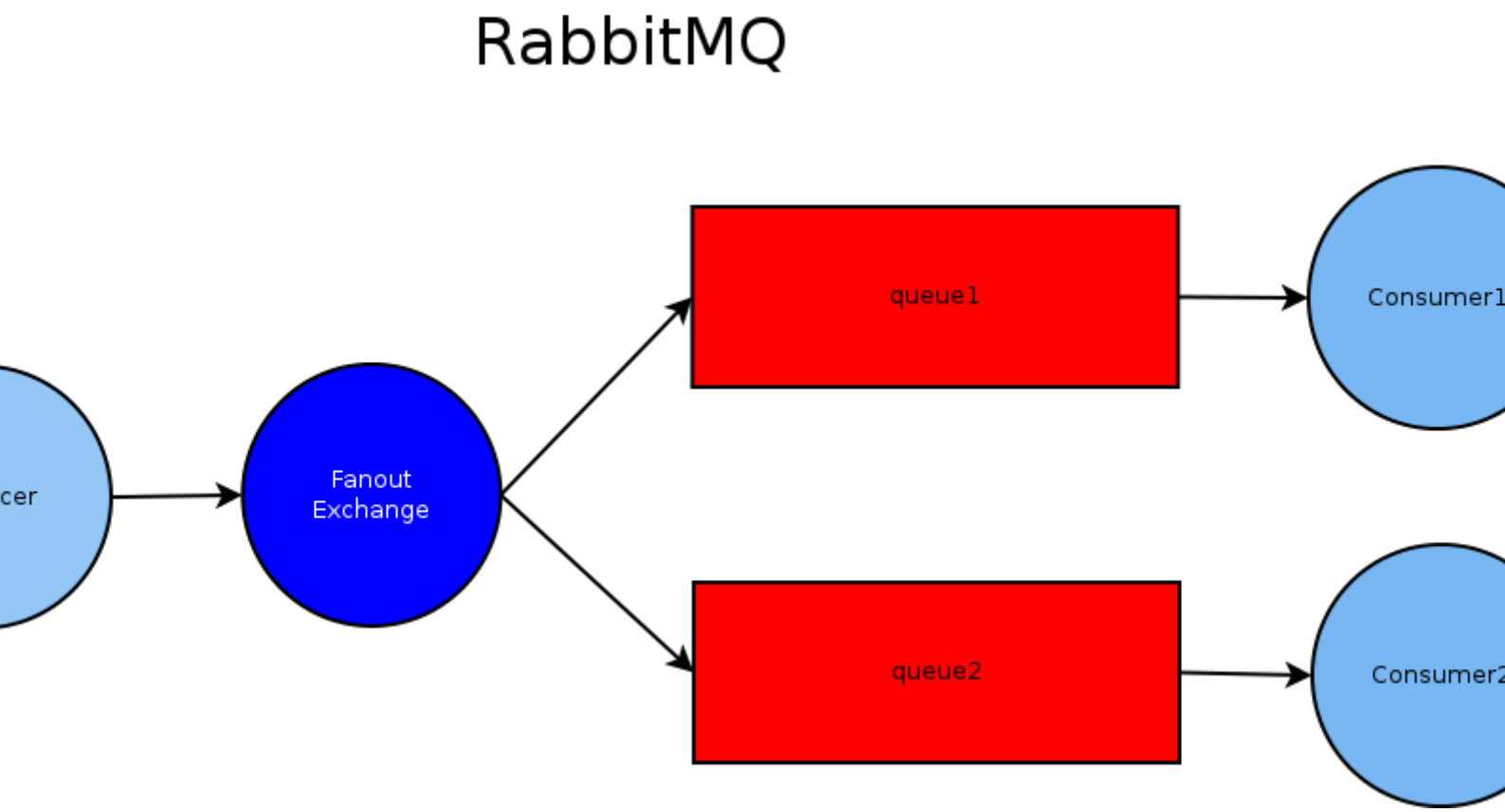
62. AMQP

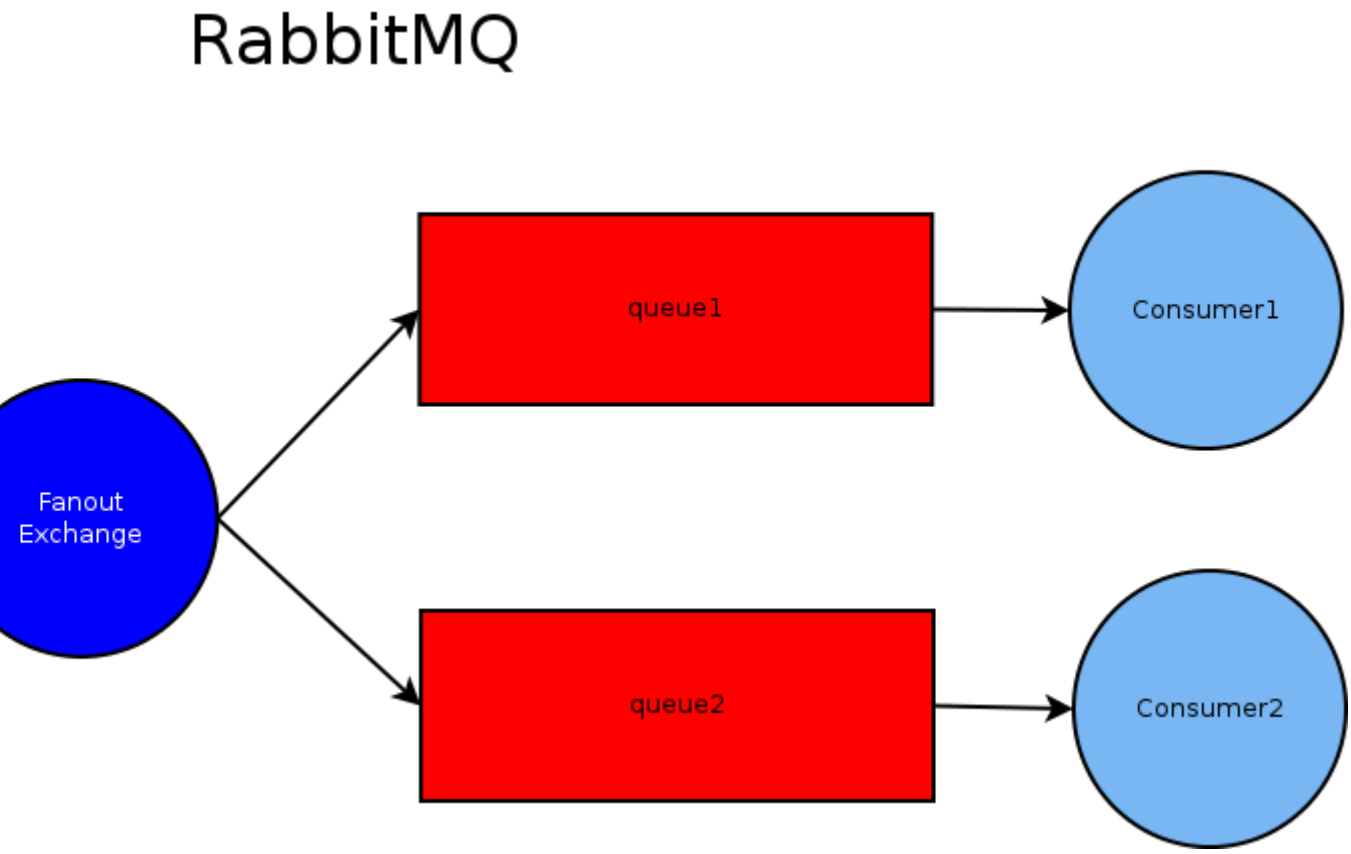
- **Protokoł warstwy połączenia (wire-level protocol)**

JMS → API

- **Wieloplatformość**
- **Utrwalanie wiadomość**
- **Potwierdzenie odbioru**
- **Skalowalność**
- **Popularność**
- **Modelowanie różnych schematów kierowanie**

W JMS tylko punkt-punkt oraz publikacja-subskrypcja





62.1. Wymiana komunikatów

- **Producent**

Wysyła wiadomość do exchange.

NOTE : Producent nie publikuje

- **Wymiana (exchange)**

Kieruje komunikaty do odpowiedniej kolejki lub kolejek

Routing agent

Trasowanie i filtrowanie komunikatów

- Może być trafały - przeżyje restart brokera
- Może być tymczasowy
- Może być auto-kasowalny - skasowany jeśli żadna kolejka nie będzie z nim dłużej powiązana
  - Kolejka
  - Konsument

62.2. Komponenty

- **Kanał**

logiczna reprezentacja połączenia

- **Binding**

relacja pomiędzy queue a exchange

- **Message**

Wiadomość

- **Broker**

Zarządza exchange'ami , kolejkami etc

- **Kolejka**

Odbiera wiadomość wysłaną z exchange

62.3. Exchange Type

- **direct**

```
Komunikat zostaje bezpośrednio skierowany do kolejki jeśli klucz trasowania zgadza
się z kluczem w powiązaniu
```

- **fanout**

wszystkie wiadomości trafiają do wszystkich kolejek, routing key dowiązania jest ignorowany

Loadbalancing

- **topic**

```
Komunikat zostaje bezpośrednio skierowany do kolejki jeśli klucz trasowania zgadza
się z kluczem w powiązaniu przy wykorzystaniu dopasowania z użyciem wzorców
wieloznacznych
```

- **headers**

Komunikat zostaje bezpośrednio skierowany do kolejki jeśli nagłówki i wartości umieszczone w tablicy argumentów będą odpowiadać nagłówkom i wartościom dostępnym w tablicy argumentów powiązania

```
spring-boot-starter-amqp
```

powoduje to załadowanie jara : **org.springframework.amqp:spring-rabbit:{someVersion}.RELEASE**

Przykład tworzenie połączenia i wywołania klienta imperatywnie :

```
ConnectionFactory connectionFactory = new CachingConnectionFactory();
AmqpAdmin admin = new RabbitAdmin(connectionFactory);
admin.declareQueue(new Queue("myqueue"));
AmqpTemplate template = new RabbitTemplate(connectionFactory);
template.convertAndSend("myqueue", "foo");
String foo = (String) template.receiveAndConvert("myqueue");
```

lub XML :

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:rabbit="http://www.springframework.org/schema/rabbit"
       xsi:schemaLocation="http://www.springframework.org/schema/rabbit
           http://www.springframework.org/schema/rabbit/spring-rabbit.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <rabbit:connection-factory id="connectionFactory"/>

    <rabbit:template id="amqpTemplate" connection-factory="connectionFactory"/>

    <rabbit:admin connection-factory="connectionFactory"/>

    <rabbit:queue name="myqueue"/>

</beans>
```

```
ApplicationContext context = new GenericXmlApplicationContext("classpath:/rabbit-
context.xml");
AmqpTemplate template = context.getBean(AmqpTemplate.class);
template.convertAndSend("myqueue", "foo");
String foo = (String) template.receiveAndConvert("myqueue");
```

lub JavaConfig :

```
@Configuration
public class RabbitConfiguration {

    @Bean
    public ConnectionFactory connectionFactory() {
        return new CachingConnectionFactory("localhost");
    }

    @Bean
    public AmqpAdmin amqpAdmin() {
        return new RabbitAdmin(connectionFactory());
    }

    @Bean
    public RabbitTemplate rabbitTemplate() {
        return new RabbitTemplate(connectionFactory());
    }

    @Bean
    public Queue myQueue() {
       return new Queue("myqueue");
    }
}
```

62.4. RabbitMQ support

```
RabbitMQ jest popularnym, otwartym brokerem komunikatów implementującym AMQP.
```

```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=admin
spring.rabbitmq.password=secret
```

Czysty Spring :

- **Tworzenie connectionFactory**

```
<bean id="connectionFactory"
      class="org.springframework.amqp.rabbit.connection.CachingConnectionFactory">
    <constructor-arg value="somehost"/>
    <property name="username" value="guest"/>
    <property name="password" value="guest"/>
    <property name="channelCacheSize" value="50"/>
</bean>
```

lub z namespace:

```
<rabbit:connection-factory id="connectionFactory" channel-cache-size="50"/>

<rabbit:connection-factory  id="connectionFactory" addresses="host1:5672,host2:5672"/>

<rabbit:connection-factory  id="connectionFactory" host="somehost" port="5672"/>
```

- **Retry (Usługa ponawiania)**

```
<rabbit:template id="template" connection-factory="connectionFactory" retry-
template="retryTemplate"/>

<bean id="retryTemplate" class="org.springframework.retry.support.RetryTemplate">
    <property name="backOffPolicy">
        <bean class="org.springframework.retry.backoff.ExponentialBackOffPolicy">
            <property name="initialInterval" value="500" />
            <property name="multiplier" value="10.0" />
            <property name="maxInterval" value="10000" />
        </bean>
    </property>
</bean>
```

lub JavaConfig:

```
@Bean
public AmqpTemplate rabbitTemplate();
    RabbitTemplate template = new RabbitTemplate(connectionFactory());
    RetryTemplate retryTemplate = new RetryTemplate();
    ExponentialBackOffPolicy backOffPolicy = new ExponentialBackOffPolicy();
    backOffPolicy.setInitialInterval(500);
    backOffPolicy.setMultiplier(10.0);
    backOffPolicy.setMaxInterval(10000);
    retryTemplate.setBackOffPolicy(backOffPolicy);
    template.setRetryTemplate(retryTemplate);
    return template;
}
```

lub bezpośrednio w kodzie :

```
retryTemplate.execute(
    new RetryCallback<Object, Exception>() {

        @Override
        public Object doWithRetry(RetryContext context) throws Exception {
            context.setAttribute("message", message);
            return rabbitTemplate.convertAndSend(exchange, routingKey, message);
        }
    }, new RecoveryCallback<Object>() {

        @Override
        public Object recover(RetryContext context) throws Exception {
            Object message = context.getAttribute("message");
            Throwable t = context.getLastThrowable();
            // Do something with message
            return null;
        }
    });
}
```

- **Wysyłanie wiadomości**

```
void send(Message message) throws AmqpException;

void send(String routingKey, Message message) throws AmqpException;

void send(String exchange, String routingKey, Message message) throws AmqpException;
```

```
amqpTemplate.send("marketData.topic", "quotes.nasdaq.FOO", new
Message("12.34".getBytes(), someProperties));
```

- **RabbitTemplate**

```
RabbitTemplate template = new RabbitTemplate(); // using default no-name Exchange
template.send("queue.helloWorld", new Message("Hello World".getBytes(),
someProperties));

//

RabbitTemplate template = new RabbitTemplate(); // using default no-name Exchange
template.setRoutingKey("queue.helloWorld"); // but we'll always send to this Queue
template.send(new Message("Hello World".getBytes(), someProperties));
```

- **MessageBuilder**

```
Message message = MessageBuilder.withBody("foo".getBytes())
    .setContentType(MessageProperties.CONTENT_TYPE_TEXT_PLAIN)
    .setMessageId("123")
    .setHeader("bar", "baz")
    .build();

//

MessageProperties props = MessagePropertiesBuilder.newInstance()
    .setContentType(MessageProperties.CONTENT_TYPE_TEXT_PLAIN)
    .setMessageId("123")
    .setHeader("bar", "baz")
    .build();

Message message = MessageBuilder.withBody("foo".getBytes())
    .andProperties(props)
    .build();
```

- **Odbieranie wiadomości**

```
Message receive() throws AmqpException;

Message receive(String queueName) throws AmqpException;

Message receive(long timeoutMillis) throws AmqpException;

Message receive(String queueName, long timeoutMillis) throws AmqpException;
```

```
boolean received =
        this.template.receiveAndReply(ROUTE, new ReceiveAndReplyCallback<Order,
Invoice>() {

                public Invoice handle(Order order) {
                        return processOrder(order);
                }
        });
if (received) {
        log.info("We received an order!");
}
```

- **Asynchronicznie**

```
public interface MessageListener {
    void onMessage(Message message);
}
```

- **Kontener**

```
SimpleMessageListenerContainer container = new SimpleMessageListenerContainer();
container.setConnectionFactory(rabbitConnectionFactory);
container.setQueueNames("some.queue");
container.setMessageListener(new MessageListenerAdapter(somePojo));
```

lub XML :

```
<rabbit:listener-container connection-factory="rabbitConnectionFactory">
    <rabbit:listener queues="some.queue" ref="somePojo" method="handle"/>
</rabbit:listener-container>


<rabbit:listener-container connection-factory="rabbitConnectionFactory">
    <rabbit:listener queues="some.queue" ref="somePojo" method="handle" priority="10"
/>
</rabbit:listener-container>
```

lub JavaConfig:

```
@Configuration
public class ExampleAmqpConfiguration {

    @Bean
    public SimpleMessageListenerContainer messageListenerContainer() {
        SimpleMessageListenerContainer container = new
SimpleMessageListenerContainer();
        container.setConnectionFactory(rabbitConnectionFactory());
        container.setQueueName("some.queue");
        container.setMessageListener(exampleListener());
        return container;
    }

    @Bean
    public ConnectionFactory rabbitConnectionFactory() {
        CachingConnectionFactory connectionFactory =
            new CachingConnectionFactory("localhost");
        connectionFactory.setUsername("guest");
        connectionFactory.setPassword("guest");
        return connectionFactory;
    }

    @Bean
    public MessageListener exampleListener() {
        return new MessageListener() {
            public void onMessage(Message message) {
                System.out.println("received: " + message);
            }
        };
    }
}
```

- **AutoListener @RabbitListener**

Aktywacja : @EnableRabbit

```
@Configuration
@EnableRabbit
public class AppConfig {

    @Bean
    public SimpleRabbitListenerContainerFactory rabbitListenerContainerFactory() {
        SimpleRabbitListenerContainerFactory factory = new
SimpleRabbitListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory());
        factory.setConcurrentConsumers(3);
        factory.setMaxConcurrentConsumers(10);
        return factory;
    }
}
```

lub XML :

```
<rabbit:annotation-driven/>

<bean id="rabbitListenerContainerFactory"

class="org.springframework.amqp.rabbit.config.SimpleRabbitListenerContainerFactory">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="concurrentConsumers" value="3"/>
    <property name="maxConcurrentConsumers" value="10"/>
</bean>
```

Listener : @RabbitListener

```
@Component
public class MyService {

    @RabbitListener(queues = "myQueue")
    public void processOrder(String data) {
        ...
    }

}
//
@Component
public class MyService {

  @RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = "myQueue", durable = "true"),
        exchange = @Exchange(value = "auto.exch", ignoreDeclarationExceptions =
"true"),
        key = "orderRoutingKey")
  )
  public void processOrder(String data) {
    ...
  }

  @RabbitListener(bindings = @QueueBinding(
        value = @Queue,
        exchange = @Exchange(value = "auto.exch"),
        key = "invoiceRoutingKey")
  )
  public void processInvoice(String data) {
    ...
  }

}
```

62.5. Wysyłanie wiadomości

```
@Component
public class MyBean {

    private final AmqpAdmin amqpAdmin;
    private final AmqpTemplate amqpTemplate;

    @Autowired
    public MyBean(AmqpAdmin amqpAdmin, AmqpTemplate amqpTemplate) {
        this.amqpAdmin = amqpAdmin;
        this.amqpTemplate = amqpTemplate;
    }

    // ...

}
```

62.6. Odbieranie wiadomości

```
@Component
public class MyBean {

    @RabbitListener(queues = "someQueue")
    public void processMessage(String content) {
        // ...
    }

}
```

62.7. Własny RabbitListenerContainerFactory

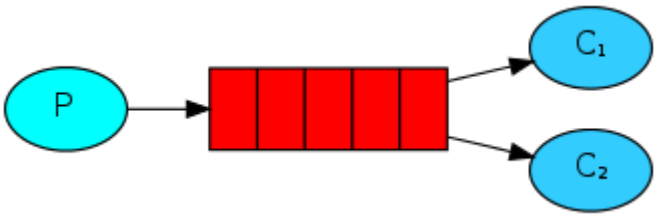
```
@Configuration
static class RabbitConfiguration {

    @Bean
    public SimpleRabbitListenerContainerFactory myFactory(
            SimpleRabbitListenerContainerFactoryConfigurer configurer) {
        SimpleRabbitListenerContainerFactory factory =
                new SimpleRabbitListenerContainerFactory();
        configurer.configure(factory, connectionFactory);
        factory.setMessageConverter(myMessageConverter());
        return factory;
    }

}
```

Wykorzytanie :

```
@Component
public class MyBean {

    @RabbitListener(queues = "someQueue", containerFactory="myFactory")
    public void processMessage(String content) {
        // ...
    }

}
```

63. Przykłady

63.1. Równoległy odbiorca

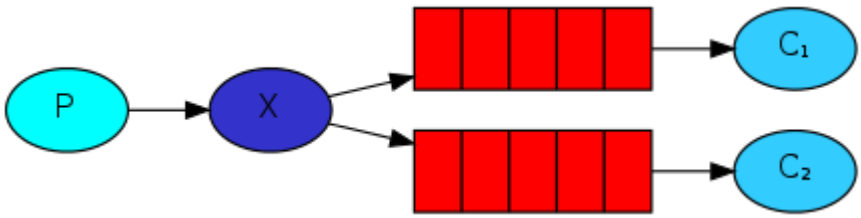


```
@RabbitListener(queues = Rabbit3Config.QUEUE_HELLO)
@Slf4j
public class Receiver3 {

    private final int instance;
    private Random random;

    public Receiver3(int instance) {
        super();
        this.instance = instance;
        random = new Random();
    }

    @RabbitHandler
    public void receive(String in) throws InterruptedException {
        StopWatch watch = new StopWatch();
        watch.start();
        log.info("instance  {} receive {}", this.instance, in);
        doWork();
        watch.stop();
        log.info("instance {} ,  Done in {} ", this.instance,
watch.getTotalTimeSeconds());
    }

    private void doWork() throws InterruptedException {
        Thread.sleep(instance * random.nextInt(2000));
    }
}
```

```
@Slf4j
public class Sender3 {

    @Autowired
    private RabbitTemplate template;

    @Autowired
    private Queue queue;

    @Scheduled(fixedDelay = 1000, initialDelay = 500)
    public void send() {
        String uuid = UUID.randomUUID().toString();
        template.convertAndSend(queue.getName(), uuid);
        log.info("Sent  {}" , uuid);
}
}
```

```
@Configuration
public class Rabbit3Config {

    public static final String QUEUE_HELLO = "helloQueue2";

    @Bean
    public Queue hello() {
        return new Queue(QUEUE_HELLO);
    }

    @Bean
    public Sender3 sender() {
        return new Sender3();
    }

    @Bean
    public Receiver3 receiveOne() {
        return new Receiver3(2);
    }

    @Bean
    public Receiver3 receiveTwo() {
        return new Receiver3(3);
    }

}
```

63.2. Pub/Sub



Konfiguracja:

```
  @Bean
    public Queue hello1() {
        return new Queue(QUEUE_HELLO1);
    }

    @Bean
    public Queue hello2() {
        return new Queue(QUEUE_HELLO2);
    }

    @Bean
    public ReceiverOne receiveOne() {
        return new ReceiverOne();
    }

    @Bean
    public ReceiverTwo receiveTwo() {
        return new ReceiverTwo();
    }
    @Bean
    public Sender sender(){
        return new Sender();
    }

    @Bean
    public FanoutExchange fanoutExchangeA(){
        return new FanoutExchange("exchange-example-my");
    }

    @Bean
    public Binding binding1(){
        return BindingBuilder.bind(hello1()).to(fanoutExchangeA());
    }

    @Bean
    public Binding binding2(){
        return BindingBuilder.bind(hello2()).to(fanoutExchangeA());
    }
```

Sender:

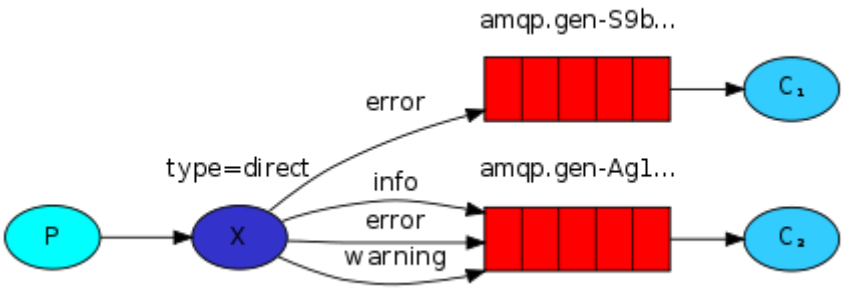
```
@Slf4j
public class Sender {
    @Autowired
    private RabbitTemplate template;
    @Scheduled(fixedDelay = 1000, initialDelay = 500)
    public void send() {
        template.setExchange("exchange-example-my");
        template.convertAndSend("Fanout hello");
    }

}
```

Receiver:

```
@Slf4j
@RabbitListener(queues = Rabbit4Config.QUEUE_HELLO2)
public class ReceiverOne {

    @RabbitHandler
    public void receive(String in) {
        log.info("Received one : {}",in );
    }

}
```

63.3. Routing

source: https://habrahabr.ru/post/262069/



Konfiguracja:

```
@Configuration
public class RoutingConfig {

    public static final String QUEUE_HELLO1 = "Queue1";
    public static final String QUEUE_HELLO2 = "Queue2";
```

```
    @Bean
    public Queue hello1() {
        return new Queue(QUEUE_HELLO1);
    }

    @Bean
    public Queue hello2() {
        return new Queue(QUEUE_HELLO2);
    }


    @Bean
    public DirectExchange directExchange(){
        return new DirectExchange("routing-exchange");
    }

    @Bean
    public Binding errorBinding1(){
        return BindingBuilder.bind(hello1()).to(directExchange()).with("error");
    }

    @Bean
    public Binding errorBinding2(){
        return BindingBuilder.bind(hello2()).to(directExchange()).with("error");
    }

    @Bean
    public Binding infoBinding(){
        return BindingBuilder.bind(hello1()).to(directExchange()).with("info");
    }

    @Bean
    public Binding warningBinding(){
        return BindingBuilder.bind(hello2()).to(directExchange()).with("warning");
    }


    @Bean
    public Receiver receiveOne() {
        return new Receiver();
    }

    @Bean
    public Receiver2 receiveTwo() {
        return new Receiver2();
    }

    @Bean
    public Sender sender(){
        return new Sender();
    }
```

Sender:

```
@Slf4j
public class Sender {
    @Autowired
    private RabbitTemplate template;
    @Scheduled(fixedDelay = 1000, initialDelay = 500)
    public void send() {

        template.setExchange("routing-exchange");
        template.convertAndSend("error", "direct message !");
        log.info("Sent :{}", "direct message");
    }

}
```

64. Actuator

Parametry życiowe i wydajnościowe instancji

64.1. Zależność

spring-boot-actuator

64.2. Kanały

- HTTP
  - http endpoints jak : /actuator, /info, /health etc
  - Jolokia

Zależność :

```
<dependency>
    <groupId>org.jolokia</groupId>
    <artifactId>jolokia-core</artifactId>
 </dependency>
```

Endpoint :

```
 /jolokia
```

Dezaktywacja :

```
endpoints.jolokia.enabled=false
```

- JMX
  - Monitoring i zarządzanie przez JMX

    ```
    Domyślnie Spring Boot tworzy instancje MBeanServer i wystawia interfejsy
    @adnotowane przez
    ```

    - @ManagedResource
    - @ManagedAttribute
    - @ManagedOperation
- SSH
  - remote shell

64.3. Co możemy zrobić z actuator endpoints ?

- Zmienić nazwę
- uaktywnić/deaktywować endpoint

```
endpoints:
  enabled: false
metrics:
  enabled: true
```

```
endpoints.beans.id=springbeans
endpoints.beans.sensitive=false
endpoints.beans.enabled=true
```

- Zdefiniować swoje własne metryki
- Stworzyć własne repozytorium do przechowywania danych
- Stworzyć swoje własne identyfikatory

```
@Component
public class CustomEndpoint implements Endpoint<List<String>> {

    @Override
    public String getId() {
        return "myEndpoint";
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public boolean isSensitive() {
        return true;
    }

    @Override
    public List<String> invoke() {
        List<String> messages = new ArrayList<>();
        messages.add("test one");
        messages.add("test 2");
        return messages;
    }
}
```

64.4. HealthEndpoint

Aktywne to :

- **CassandraHealthIndicator**
- **DiskSpaceHealthIndicator**
- **DataSourceHealthIndicator**
- **ElasticsearchHealthIndicator**
- **JmsHealthIndicator**
- **MailHealthIndicator**
- **MongoHealthIndicator**
- **RabbitHealthIndicator**
- **RedisHealthIndicator**
- **SolrHealthIndicator**

64.5. Deaktywacja wszystkich domyślnych

```
management.health.defaults.enabled
```

64.6. Sortowanie

management.health.status.order=DOWN, OUT_OF_SERVICE, UNKNOWN, UP

64.7. Własny Health endpoint / HealthIndicator

```
@Component
public class ActiveMQHealth implements HealthIndicator {
private ConnectionFactory factory;
@Autowired
public ActiveMQHealth(ConnectionFactory factory) {
   this.factory = factory;
}
@Override
public Health health() {
    try {
      factory.createConnection();
    } catch (JMSException e) {
      return new Health.Builder().down(e).build();
    }
 return new Health.Builder().status(Status.UP + ": Successfully connected tothe
broker").build();
}
}
```

```
@Component
public class MyHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        int errorCode = check(); // perform some specific health check
        if (errorCode != 0) {
            return Health.down().withDetail("Error Code", errorCode).build();
        }
        return Health.up().build();
    }

}
```

- **/autoconfig** - wyświetla co zostało skonfigurowane automatycznie
- **/beans** - wyświetla wszystkie bean'y zarejestrowane w aplikacji

- **/configprops** - wszystkie konfiguracje properties
- **/dump** - dump report
- **/env** - report o bieżących ustawieniach systemowych
  - /env/{key} przykład http://localhost:9991/env/path.separator - pobranie separatora
- **/health** - prosty raport funkcji życiowych aplikacji

```
curl localhost:8090/health      h
{"status":"UP","diskSpace":{"status":"UP","total":219353915392,"free":127528636416,"th
reshold":10485760},"mongo":{"status":"UP","version":"2.6.10"}}
```

- **/info** - serwuje informacje o bieżącej konfiguracji aplikacji
  - Parametryzacja / prefix info.

info.name=App test info.project=Camp Spring Boot info.manuscript.raw=asciidoctor info.desc=A Little introduce to Spring Boot world

- Wynik :

```
{

    "name": "App test",
    "build": {
        "gradleVersion": "2.14",
        "artifact": "kataBoot",
        "version": "0.0.1-SNAPSHOT"
    },
    "desc": "A Little introduce to Spring Boot world",
    "project": "Camp Spring Boot",
    "manuscript": {
        "raw": "asciidoctor"
    }

}
```

64.8. InfoContributors

- **EnvironmentInfoContributor** - info ze środowiska
- **GitInfoContributor** = jeśli plik git.properties jest dostępny
  - git.branch
  - git.commit.id
  - git.commit.time
- **BuildInfoContributor** = informacje z META-INF/build-info.properties jeśli plik jest dostępny

64.9. Zmienne aktywne w czasie budowanie projektu

```
info.app.encoding=@project.build.sourceEncoding@
info.app.java.source=@java.version@
info.app.java.target=@java.version@
```

64.9.1. Własna implementacja

- Przykład

```
 @Bean
   InfoEndpoint info(){
       Map<String,String> info = Maps.newHashMap();
       info.put("slawek", "borowiec");
       return new InfoEndpoint(info);
   }
```

- Przykład 2

```
  @Autowired
    private ConfigurableEnvironment environment;


    @Bean
   InfoEndpoint info() throws Exception{
        MutablePropertySources propertySource = environment.getPropertySources();
        PropertiesConfigurationFactory factory = new
PropertiesConfigurationFactory(new HashMap<String, Object>());
        factory.setTargetName("info");
        factory.setPropertySources(propertySource);
        Map<String,Object> info = new HashMap<>();
        info.put("info", factory.getObject());
        return new InfoEndpoint(info);
   }
```

- **/metrics** - metryki dotyczące punktów końcowych, sterty, wątków, gc, class loading etc

Dodatkowo dla każdej strony trzyma statusy HTTP, licznik wywołań, gauge.response

```
{

    "mem": 1132031,
    "mem.free": 485640,
    "processors": 8,
    "instance.uptime": 1079395,
    "uptime": 1088537,
    "systemload.average": 1.09,
    "heap.committed": 1048576,
    "heap.init": 1048576,
    "heap.used": 561152,
    "heap": 2097152,
    "nonheap.committed": 84928,
    "nonheap.init": 2496,
    "nonheap.used": 83457,
    "nonheap": 0,
    "threads.peak": 42,
    "threads.daemon": 38,
    "threads.totalStarted": 79,
    "threads": 42,
    "classes": 10148,
    "classes.loaded": 10148,
    "classes.unloaded": 0,
    "gc.g1_young_generation.count": 4,
    "gc.g1_young_generation.time": 141,
    "gc.g1_old_generation.count": 0,
    "gc.g1_old_generation.time": 0,
    "gauge.servo.response.health": 8.0,
    "gauge.servo.rest.totaltime": 0.008942816666666667,
    "gauge.servo.rest.count": 0.016666666666666666,
    "gauge.servo.rest.min": 1.155877,
    "gauge.servo.rest.max": 1.155877,
    "gauge.servo.response.api.applications": 3.0,
    "httpsessions.max": -1,
    "httpsessions.active": 0

}
```


- /**mappings** - wszystkie mapowania URL w aplikacji
- /**trace** - detale wcześniejszy requestów


```
{

    "timestamp": 1468504478026,
    "info": {
        "method": "POST",
        "path": "/api/applications",
        "headers": {
            "request": {
```

```
                "accept": "application/json",
                "content-type": "application/json",
                "user-agent": "Java/1.8.0_91",
                "host": "localhost:8090",
                "connection": "keep-alive",
                "content-length": "273"
            },
            "response": {
                "X-Application-Context": "CEP:dev:8090",
                "Content-Type": "application/json;charset=UTF-8",
                "Transfer-Encoding": "chunked",
                "Date": "Thu, 14 Jul 2016 13:54:38 GMT",
                "status": "201"
            }
        }
    }

},
{

    "timestamp": 1468504468058,
    "info": {
        "method": "GET",
        "path": "/api/journal",
        "headers": {
            "request": {
                "host": "localhost:8090",
                "user-agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0)
Gecko/20100101 Firefox/44.0",
                "accept": "text/event-stream",
                "accept-language": "pl,en-US;q=0.7,en;q=0.3",
                "accept-encoding": "gzip, deflate",
                "referer": "http://localhost:8090/index.html",
                "cookie":
"__utma=111872281.2131442542.1458644832.1458644832.1461744507.2;
__utmz=111872281.1458644832.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none);
_ga=GA1.1.2131442542.1458644832; auth_token=5b97a1d44bf087ab990369e6d6e2d55c413994d4;
JSESSIONID=F70300B2C0A464FA47C28817326F8988",
                "connection": "keep-alive",
                "pragma": "no-cache",
                "cache-control": "no-cache"
            },
            "response": {
                "X-Application-Context": "CEP:dev:8090",
                "Content-Type": "text/event-stream;charset=UTF-8",
                "Transfer-Encoding": "chunked",
                "Date": "Thu, 14 Jul 2016 13:54:28 GMT",
                "status": "200"
            }
        }
    }
```

```
},...
```

- Przykład

/trace domyślnie przechowuje 100 wpisów jeśli chcemy to zwiększyć należy skorzystać z kodu poniżej

```
@Configuration
public class ActuatorConfig {
@Bean
public InMemoryTraceRepository traceRepository() {
InMemoryTraceRepository traceRepo = new InMemoryTraceRepository();
traceRepo.setCapacity(1000);
return traceRepo;
}
}
```

- Utrwalanie wyników w mongo

```
@Service
public class MongoTraceRepository implements TraceRepository {
private MongoOperations mongoOps;

@Autowired
public MongoTraceRepository(MongoOperations mongoOps) {
this.mongoOps = mongoOps;
}

@Override
public List<Trace> findAll() {
  return mongoOps.findAll(Trace.class);
}
@Override
public void add(Map<String, Object> traceInfo) {
   mongoOps.save(new Trace(new Date(), traceInfo));
}

}
```

64.10. remote shell

```
spring-boot-starter-remote-shell
```

- Polecenie : **ssh -p 2000 user@localhost**

- Parametryzowanie wtyczki :

```
shell.ssh.port
shell.auth.simple.user.name
shell.auth.simple.user.password
```

64.11. Własne metryki

- CounterService

```
...
final private CounterService counterService;
counterService.increment("messages.total.book.added");
```

- increment
- decrement
- reset
    - GaugeService

Zapisuje pojedyńczą własność oraz jej parametr

```
public interface GaugeService {
  void submit(String metricName, double value);
}
```

- Przykład

```
@Controller
@RequestMapping("/person")
@ConfigurationProperties("person")
public class PersonController {
...
private CounterService counterService;
@Autowired

public ReadingListController(PersonRepository personRepository,PersonProperties
personProperties,
CounterService counterService, GaugeService gaugeService) {
....
}
...
@RequestMapping(method=RequestMethod.POST)
public String addNewInvoiceToPerson(@Valid Invoice invoice, @Valid Long personId) {

personRepository.findBy(personId).addInvoice(invoice);

counterService.increment("invoice.saved");
gaugeService.submit("invoice.last.saved", System.currentTimeMillis());
return "redirect:/";
}
}
```

64.12. Trwałe metryki

- MetricRepository - programowy dostęp do metryk
    - Metryki w JMX

Przykład:

```
 @Bean
    @ExportMetricWriter
    MetricWriter metricWriter(MBeanExporter exporter) {
        return new JmxMetricWriter(exporter);
    }

    @Bean
    @Primary
    MBeanExporter mBeanExporter() {
        return new MBeanExporter();
}
```

- MetricReader
- MetricWriter

Dostarczene przez : **InMemoryMetricRepository**, **RedisMetricRepository**

- Przykład 1

```
   @Bean
InMemoryMetricRepository inMemoryMetricRepository() {
    return new InMemoryMetricRepository();
}
```

- Przykład 2

```
@Autowired
private ApplicationContext context;

        @Bean
    protected MetricWriter writer() {
        RedisMetricRepository repository = new
RedisMetricRepository(connectionFactory);
        repository.setPrefix("spring.metrics.collector." + context.getId() + "." +
ObjectUtils.getIdentityHexString(context) + ".");
        repository.setKey("keys.spring.metrics.collector");
        return repository;
}

        @Bean
    @Primary
    protected MetricRepository reader() {
        return new InMemoryMetricRepository();
}

//redis setting connection


@Bean(name="org.springframework.autoconfigure.redis.RedisAutoConfiguration$RedisProper
ties")
    @Primary
    public RedisProperties redisProperties() throws UnknownHostException {
        RedisProperties properties = new RedisProperties();
        try {
            CloudFactory cloud = new CloudFactory();
            RedisServiceInfo serviceInfo = (RedisServiceInfo) cloud.getCloud()
                    .getServiceInfo("redis");
            if (serviceInfo != null) {
                properties.setHost(serviceInfo.getHost());
                properties.setPassword(serviceInfo.getPassword());
                properties.setPort(serviceInfo.getPort());
            }
        } catch (CloudException e) {
            // ignore
        }
        return properties;
}
```

```
@Bean
    @ExportMetricWriter
    MetricWriter metricWriter(MetricExportProperties export) {
        return new RedisMetricRepository(connectionFactory,
             export.getRedis().getPrefix(), export.getRedis().getKey());
}
```

64.13. Dostrajanie portów

```
management.port=9991
management.address=127.0.0.1
management.context-path=/manage
```

- Tylko JMX

```
management.port=-1
```

- Wyłączenie JMX

```
endpoints.jmx.enabled=false
```

64.14. Zmiana nazwy endpointu

endpoints.info.id=campInfo

64.15. Rozszerzenia :

- Dropwizard
- io.dropwizard.metrics:metrics-graphite

64.16. Zamykanie aplikacji

curl -X POST http://localhost:8080/shutdown

```
endpoints:
shutdown:
enabled: true
```

64.17. Zabezpiecznie / Security

```
@Override
protected void configure(HttpSecurity http) throws Exception {
http
    .authorizeRequests()
    .antMatchers("/").access("hasRole('USER')")
    .antMatchers("/info").access("hasRole('ROOT')")
    .antMatchers("/metrics").access("hasRole('ROOT')")
        ...
    .antMatchers("/shutdown").access("hasRole('ADMIN')")
    .antMatchers("/**").permitAll()
    .and()
    .formLogin()
    .loginPage("/login")
    .failureUrl("/login?error=true");
}
```

```
management.context-path=/reports


.antMatchers("/reports/**").access("hasRole('ADMIN')")
```

64.18. Security / denial of service

endpoints.health.time-to-live=1000 //ms

64.19. Spring Boot Admin

Zależność :

```
    compile group: 'de.codecentric', name: 'spring-boot-admin-server', version:
'1.3.4'
    compile group: 'de.codecentric', name: 'spring-boot-admin-server-ui', version:
'1.3.4'
    compile group: 'de.codecentric', name: 'spring-boot-admin-starter-client',
version: '1.3.4'
```

Konfiguracja

```
@Configuration
public class AdminBootConfig extends WebMvcConfigurerAdapter{

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("index.html").addResourceLocations("classpath:/META-
INF/spring-boot-admin-server-ui/");
    }
}
```

Aktywacja :

```
@SpringBootApplication
@EnableAdminServer
public class MyBooApp   {
        public static void main(String[] args)  {
                SpringApplication.run(MyBootApp.class,  args);
        }
}
```

application.properties

```
server.port:8090
spring.application.name:Spring Boot Admin Web
spring.boot.admin.url:http://localhost:${server.port}
```

64.20. Dropwizard

Zależności :

```
compile group: 'io.dropwizard.metrics', name: 'metrics-core', version: '3.1.2'
compile group: 'io.dropwizard.metrics', name: 'metrics-jvm', version: '3.1.2'
compile group: 'io.dropwizard.metrics', name: 'metrics-healthchecks', version: '3.1.2'
compile group: 'io.dropwizard.metrics', name: 'metrics-logback', version: '3.1.2'
compile group: 'io.dropwizard.metrics', name: 'metrics-servlets', version: '3.1.2'
compile group: 'com.ryantenney.metrics', name: 'metrics-spring', version: '3.1.3'
```

```
@Configuration
@Slf4j
public class MetricsConfig extends MetricsConfigurerAdapter {

    @Bean(name = "metrics")
    public MetricRegistry metricRegistry() {
        final MetricRegistry metrics = new MetricRegistry();
        metrics.register("jvm.gc", new GarbageCollectorMetricSet());
        metrics.register("jvm.memory", new MemoryUsageGaugeSet());
        metrics.register("thread-states", new ThreadStatesGaugeSet());
        metrics.register("jvm.fd.usage", new FileDescriptorRatioGauge());
        return metrics;
    }

    @Value("${metrics.refresh.interval:30}")
    private long perfStatsPeriod;

    @Override
    public void configureReporters(final MetricRegistry metricRegistry) {

        registerReporter(
                Slf4jReporter.forRegistry(metricRegistry).outputTo(log).
convertRatesTo(TimeUnit.MILLISECONDS).convertDurationsTo(TimeUnit.MILLISECONDS).build(
))
                        .start(perfStatsPeriod, TimeUnit.SECONDS);

        registerReporter(JmxReporter.forRegistry(metricRegistry).build());

    }

    @Bean
    @Autowired
    public ServletRegistrationBean servletRegistrationBean(MetricRegistry
metricRegistry) {
        MetricsServlet ms = new MetricsServlet(metricRegistry);
        ServletRegistrationBean srb = new ServletRegistrationBean(ms,
"/metrics/stats/*");
        srb.setLoadOnStartup(1);
        return srb;
    }
```

65. Sending email

65.1. Zależność

```
spring-boot-starter-mail
```

Jeśli jest dostępna zależność i ustawiony parametr spring.mail.host JavaMailSender jest tworzony jeśli nie istnieje.

65.2. Parametryzacja

```
prefix : spring.mail namespace
```

66. Testing

66.1. Moduł

```
spring-boot-starter-test
```

66.2. Zależności dodane :

- Spring Test
- Spring Boot Test
- JUnit
- AssertJ
- Hamcrest
- Mockito
- JsonPath
- JSONAssert

66.3. Testy integracyjne

Przykład:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=MyApp.class, loader=SpringApplicationContextLoader.
class)
public class MyTest {
```

Przykład:

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
public class MyTest {

    // ...

}
```

Przykład:

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@IntegrationTest
public class MyTest {

}
```

Przykład:

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@WebIntegrationTest
public class MyTest {

    // ...

}
```

67. Spring 1.4 usprawnienia

67.1. @RunWith(SpringRunner.class)

Zamiennik dla SpringJUnit4ClassRunner

Przykład:

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
public class MyTest {

    // ...

}
```

Przykład:

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
public class MyTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void test() {
        this.restTemplate.getForEntity(
            "/{username}/vehicle", String.class, "Phil");
    }

}
```

67.2. Mock

Przykład:

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class SampleTestApplicationWebIntegrationTests {

    @Autowired
    private TestRestTemplate restTemplate;

    @MockBean
    private VehicleDetailsService vehicleDetailsService;

    @Before
    public void setup() {
        given(this.vehicleDetailsService.
            getVehicleDetails("123")
        ).willReturn(
            new VehicleDetails("Honda", "Civic"));
    }

    @Test
    public void test() {
        this.restTemplate.getForEntity("/{username}/vehicle",
            String.class, "sframework");
    }

}
```

67.3. @JsonTest

ObjectMapper

- **@JsonComponent**

Przykład:

```
public class VehicleDetailsJsonTests {

    private JacksonTester<VehicleDetails> json;

    @Before
    public void setup() {
        ObjectMapper objectMappper = new ObjectMappper();
        // Possibly configure the mapper
        JacksonTester.initFields(this, objectMappper);
    }

    @Test
    public void serializeJson() {
        VehicleDetails details =
            new VehicleDetails("Honda", "Civic");

        assertThat(this.json.write(details)).isEqualToJson("vehicledetails.json");

        assertThat(this.json.write(details)).hasJsonPathStringValue("@.make");

        assertThat(this.json.write(details)).extractingJsonPathStringValue("@.make")
.isEqualTo("Honda");
    }

    @Test
    public void deserializeJson() {
        String content = "{\"make\":\"Ford\",\"model\":\"Focus\"}";

        assertThat(this.json.parse(content)).isEqualTo(new VehicleDetails("Ford",
"Focus"));

        assertThat(this.json.parseObject(content).getMake()).isEqualTo("Ford");
    }

}
```

67.4. Kontrolery

```
@WebMvcTest(MyController.class)
```

- @Controller
- @ControllerAdvice
- @JsonComponent
- Filter
- WebMvcConfigurer
- HandlerMethodArgumentResolver

Dostarcza :

```
MockMvc mvc

MockMvcRequestBuilders

MockMvcResultMatchers
```

67.5. Dodanie filtrów

```
mockMvc = standaloneSetup(new PersonController()).addFilters(new
CharacterEncodingFilter()).build();
```

Przykład :

```
@RunWith(SpringRunner.class)
@WebMvcTest(UserVehicleController.class)
public class UserVehicleControllerTests {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private UserVehicleService userVehicleService;

    @Test
    public void getVehicleShouldReturnMakeAndModel() {
        given(this.userVehicleService.getVehicleDetails("sboot")).willReturn(new
VehicleDetails("Honda", "Civic"));

        this.mvc.perform(get("/car/vehicle").accept(MediaType.TEXT_PLAIN)).andExpect
(status().isOk()).andExpect(content().string("Honda Civic"));
    }

}
```

67.6. JPA

- **@DataJpaTest**
- **TestEntityManager**
- **Domyśla obsługa transakcji i wycofania**
- **Baza danych w pamięci**
- **Auto-konfiguracja Hibernate, Spring Data oraz DataSource**
- **@EntityScan**

Przykład:

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class UserRepositoryTests {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private UserRepository repository;

    @Test
    public void findByUsernameShouldReturnUser() {
        this.entityManager.persist(new User("sboot", "123"));
        User user = this.repository.findByUsername("sboot");

        assertThat(user.getUsername()).isEqualTo("sboot");
        assertThat(user.getVin()).isEqualTo("123");
    }

}
```

67.7. Testy klientów REST

Przykład:

```
@RestClientTest(MyClientService.class)
RestTemplateBuilder
MockRestServiceServer
server.expect(requestTo("/...")).andRespond(...)
```