



# Logowanie

## Łukasz Bednarski



# Praca z Loggerem

Logi aplikacji to jedna z ważniejszych rzeczy w aplikacji, które nie przekładają się bezpośrednio na funkcjonalność.

Podstawową rolą biblioteki do logowania jest przekazanie komunikatu wygenerowanego w kodzie (np. informacji, że jakąś operacja się powiodła) do odpowiedniego miejsca (np. standardowe wyjście konsoli, plik, serwer zdalny), dodając informacje o czasie, poziomie i ewentualnie kilka innych pomocniczych.



# Poziom logowania

Poziom to nic innego jak ‚ważność’ danej wiadomości, często używanym schematem są opcje:

FATAL, ERROR, WARNING, INFO, DEBUG, TRACE

w różnych wariantach (np. brak poziomu FATAL czy TRACE). Niektóre biblioteki pozwalają na definiowanie własnych poziomów (np. Log4J).

Poza informacją w logu na temat poziomu (opcjonalnie, w zależności od konfiguracji), używany jest on także w konfiguracji – np. na środowisku produkcyjnym wyłączając wszystkie logi ‚niższe’ niż INFO. Dzięki temu nie trzeba dodawać warunków i tymczasowych logów na czas rozwijania lub debugowania konkretnego fragmentu kodu, wystarczy skonfigurować odpowiednio loggery i poprawnie używać poziomów.

# Poziom logowania



- **fatal** – błędy krytyczne, nieobsłużone i blokujące działanie systemu / procesu
- **error** – błędy, które są obsługiwane i blokują tylko kreślona część procesu/wycinek funkcjonalności
- **warn** – błędy, które są obsłużone, a ich wpływ na proces został zniwelowany (np. poprzez ponowienie czynności)
- **info** – informacje do audytu, zakończenie realizacji ważnego procesu
- **debug** – informacje developerskie, np. pełne obiekty zamiast samych id.



# Dobre praktyki

1. Zwracaj uwagę na to, co logujesz!

Z reguły logi aplikacji są gorzej chronione niż np. baza danych.

2. Jeśli nie ma takiej potrzeby - nie loguj całych obiektów:

Np. tworzenie użytkownika powoduje zapisanie informacji do logu ze wszystkimi szczegółami (wywołując metodę `toString()` obiektu), przez co objętość logu rośnie dramatycznie, szczególnie w przypadku kompleksowych obiektów z wieloma poziomami zagnieżdżenia oraz rodzi to potencjalne ryzyko wycieku poufnych informacji (np. skrótu hasła, adresu zamieszkania itp)

3. Loguj krótkie i zwięzłe informacje.

W rzeczywistości w logu wystarczająca jest informacja, że został utworzony użytkownik o loginie xyz oraz id 123

4. Gorsze od braku logów jest tylko ich nadmiar.



# Java Logging API

- Pakiet `java.util.logging` jest częścią Javy SE,
- Do użycia nie jest konieczne dodawanie żadnych zależności do projektu.
- Zapewnia bardzo prosty „szkielet”, w którym większość specyficznych elementów (np. inne formatowanie, integracje z systemem zbierania logów itp) musimy samodzielnie zaprogramować.
- JUL loguje domyślnie na standardowe wyjście błędów,
- Posiada niespotykane w innych implementacjach poziomy logowania (np. „CONFIG”, „FINE”, „FINER”, które zastępują znane nam „DEBUG”).

Jest to jednak framework bardzo prosty, czasem określany wręcz naiwnym w kwestii implementacji, i może przysporzyć problemów w większych projektach, które intensywnie korzystają z loga. W podstawowej konfiguracji jest też mało wygodny w użyciu (np. ustawianie poziomu logowania). Raczej niezalecany w produkcyjnych aplikacjach.



# Java Logging API

```
LoggerExample.java
package pl.lbednarski.logger;

import java.util.logging.Logger;

public class LoggerExample
{
    protected static final Logger log = Logger.getLogger(LoggerExample.class.getName());

    public static void main( String[] args ) {
        log.info("To jest Log o poziomie Info");
        //doSomeAction();
        log.warning("To jest Log o poziomie warning");
    }
}
```

Utworzenie  
instancji loggera

Logowanie

Run	LoggerExample
	/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java .. sie 16, 2017 5:34:27 PM pl.lbednarski.logger.LoggerExample main INFO: To jest Log o poziomie Info sie 16, 2017 5:34:27 PM pl.lbednarski.logger.LoggerExample main WARNING: To jest Log o poziomie warning
	Process finished with exit code 0



# Poziom logowania

```
protected static final Logger log = Logger.getLogger(App.class.getName());  
  
public static void main( String[] args )  
{  
    log.setLevel(Level.WARNING); ←  
    log.warning("warning");  
    //doSomething();  
    log.info("info");  
}
```

Ustawienie poziomu logowania

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...  
paź 11, 2017 12:48:33 PM SDA.App main  
WARNING: warning  
  
Process finished with exit code 0
```



# Zadanie

Przygotuj prostą imitację bazy danych służącą do rejestracji i logowania użytkowników. Obiekty zapisz na liście.

1. Utwórz klasę User zawierającą kilka pól (id, imię, nazwisko, login, hasło, e-mail oraz data urodzenia).
2. Utwórz interface UserService posiadający kilka metod:
  1. Rejestracja użytkownika
  2. Logowanie użytkownika
  3. Zmiana hasła użytkownika
3. Przygotuj implementację interface'u UserService.
4. Dodaj Logger pozwalający logować informacje:
  1. Kto się zarejestrował
  2. Kto próbuje się zalogować i czy logowanie się powiodło
  3. Kto wnioskuje o zmianę hasła i czy próba się powiodła



# Log4J

Log4J mimo upływu czasu pozostaje jedną z najpopularniejszych bibliotek do logowania – całkiem słusznie, ponieważ jest naprawdę solidną implementacją z wieloma możliwościami.

Obiekty:

1. Logger - zawiera metody do tworzenia logów
2. Appender - definiuje miejsca do których trafiają logi
3. Layout - definiuje schemat wyświetlanych logów

# Zależność Maven



```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
</dependency>
```



# Wyświetlanie logów w konsoli

```
# Root logger option
log4j.rootLogger=INFO, stdout

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```



# Log4J

The screenshot shows an IDE interface with the following details:

- Project View:** Shows a project named "Logger" with the following structure:
  - .idea
  - src
    - main
      - java
        - pl.lbednarski.logger
      - resources
        - log4j.properties
    - test
  - target
  - Logger.iml
  - pom.xml
- Code Editor:** Two tabs are open: "LoggerExample.java" and "log4j.properties". The "LoggerExample.java" tab contains the following code:

```
package pl.lbednarski.logger;  
import org.apache.log4j.Logger;  
  
public class LoggerExample  
{  
    protected static final Logger log = Logger.getLogger(LoggerExample.class.getName());  
  
    public static void main( String[] args ) {  
        log.info("To jest Log o poziomie Info");  
        //doSomeAction();  
        log.warn("To jest Log o poziomie warn");  
        log.debug("To jest Log o poziomie debug");  
    }  
}
```
- Run Tab:** Shows the output of running the application. The log output is:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...  
2017-08-16 17:45:17 INFO  LoggerExample:10 - To jest Log o poziomie Info  
2017-08-16 17:45:17 WARN  LoggerExample:12 - To jest Log o poziomie warn  
2017-08-16 17:45:17 DEBUG LoggerExample:13 - To jest Log o poziomie debug  
Process finished with exit code 0
```

The screenshot shows the "Run" tab of the IDE displaying the following log output:

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...  
2017-08-16 17:45:17 INFO  LoggerExample:10 - To jest Log o poziomie Info  
2017-08-16 17:45:17 WARN  LoggerExample:12 - To jest Log o poziomie warn  
2017-08-16 17:45:17 DEBUG LoggerExample:13 - To jest Log o poziomie debug  
Process finished with exit code 0
```



# Zapis logów do pliku

```
# Root logger option
log4j.rootLogger=INFO, file

# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender

log4j.appender.file.File=C:\\loging.log
log4j.appender.file.MaxFileSize=10MB
log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```



# Zapis logów do bazy

```
# Define the root logger with appender file
log4j.rootLogger = INFO, DB

# Define the DB appender
log4j.appender.DB=org.apache.log4j.jdbc.JDBCAppender

log4j.appender.DB.URL=jdbc:mysql://localhost/DBNAME
log4j.appender.DB.driver=com.mysql.jdbc.Driver

log4j.appender.DB.user=user_name
log4j.appender.DB.password=password

# Set the SQL statement to be executed.
log4j.appender.DB.sql=INSERT INTO LOGS VALUES('%x','%d','%C','%p','%m')

# Define the layout for file appender
log4j.appender.DB.layout=org.apache.log4j.PatternLayout
```



# Zapis logów do bazy

```
CREATE TABLE LOGS
(USER_ID VARCHAR(20) NOT NULL,
DATED DATE NOT NULL,
LOGGER VARCHAR(50) NOT NULL,
LEVEL VARCHAR(10) NOT NULL,
MESSAGE VARCHAR(1000) NOT NULL
);
```

```
mysql > select * from LOGS;
+-----+-----+-----+-----+-----+
| USER_ID | DATED      | LOGGER     | LEVEL    | MESSAGE   |
+-----+-----+-----+-----+-----+
|          | 2010-05-13 | log4jExample | DEBUG    | Debug     |
|          | 2010-05-13 | log4jExample | INFO     | Info      |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

# Ciekawostka - ELK Stack



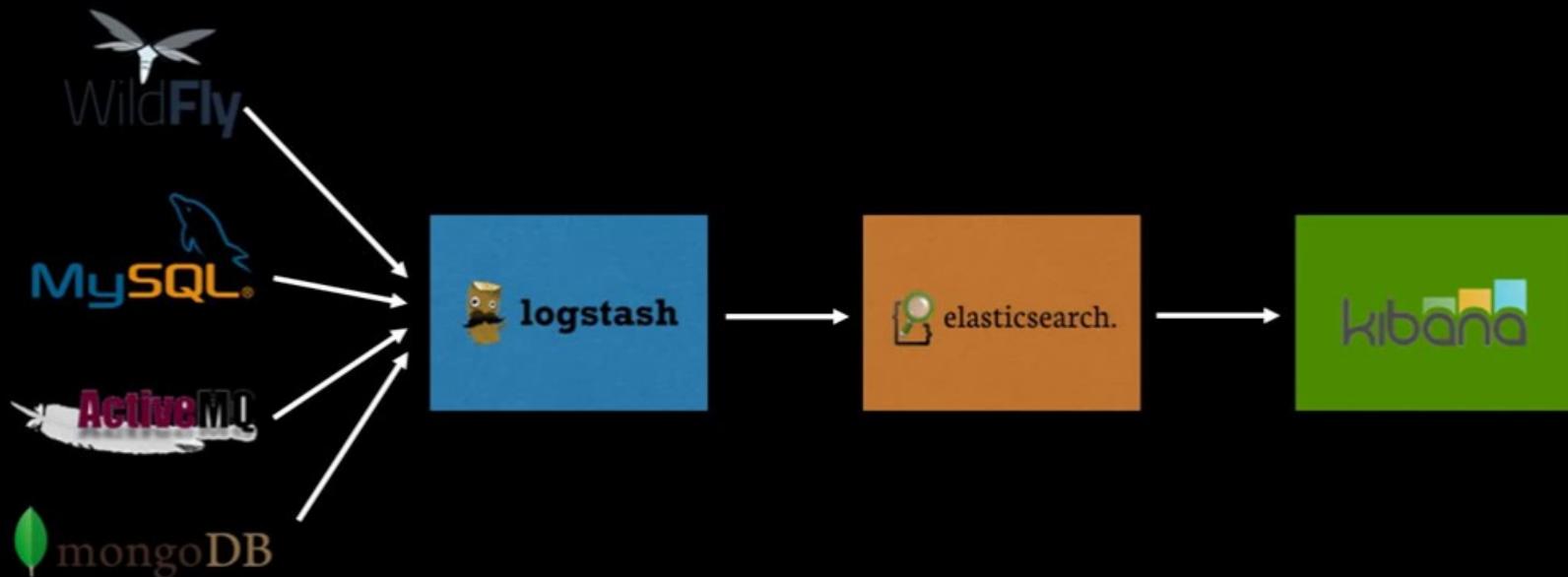
- Elasticsearch - silnik wyszukiwania pełnotekstowego oparty na Apache Lucene.
- Logstash - narzędzie przetwarzające, filtrujące, normalizujące i wysyłające logi (w naszym przypadku do Elasticsearch'a).
- Kibana - interfejs, aplikacja webowa do przeglądania i wizualizacji logów w czasie rzeczywistym.

Więcej: <http://krzysztofjelonek.net/elk-stack-dobre-praktyki-zarzadzania-logami/>

# ELK - Elastic Search / Logstash / Kibana



LOGSTASH



# Kibana



Kibana

localhost:5601/app/kibana#/discover?\_g={()&\_a=(columns:(message,method,stack\_trace),index:'logstash-\*',interval:auto,query:(query\_string:(analyzeWildcard:t.query:'\*')),sort:[(@timestamp,desc)])}

690 hits

Discover logstash-\* Selected Fields

Count December 18th 2016, 16:01:34.261 - December 18th 2016, 16:16:34.261 — by 30 seconds

Time message method stack\_trace

Time	message	method	stack_trace
December 18th 2016, 16:16:31.383	Testowy log 612	main	-
December 18th 2016, 16:16:30.382	Testowy log 611	main	-
December 18th 2016, 16:16:29.376	Testowy log 610	main	-
December 18th 2016, 16:16:29.376	Testowy error log	main	java.lang.Exception at net.krzyzstofjelonek.elktest.ELKTest.main(ELKTest.java:21)
December 18th 2016, 16:16:28.372	Testowy log 609	main	-
December 18th 2016, 16:16:27.370	Testowy log 608	main	-
December 18th 2016, 16:16:26.366	Testowy log 607	main	-
December 18th 2016, 16:16:25.356	Testowy log 606	main	-
December 18th 2016, 16:16:24.356	Testowy log 605	main	-
December 18th 2016, 16:16:23.356	Testowy log 604	main	-

Collapse

# Kibana Wizualizacja





# Zadanie

Wróć do zadania z rejestracją użytkowników.

1. Dodaj w projekcie zależność od Log4J
2. Dodaj konfigurację pozwalającą wyświetlić logi w konsoli
3. Zmień implementację programu tak, aby logi wyświetlane były za pomocą Log4J.
4. Zmodyfikuj rejestrację użytkowników w taki sposób aby rejestrowany użytkownik miał unikatowy login i e-mail.
5. Dodaj sprawdzanie wieku, użytkownicy poniżej 18 roku życia nie mogą się rejestrować.
6. Dodaj reguły sprawdzające siłę hasła (np. długość hasła minimum 8 znaków i przynajmniej 1 dużą i małą literę).
7. Utwórz 2 wyjątki:
  1. Jeśli hasło jest zbyt „słabe” np. IncorrectUserPasswordException
  2. Jeśli login lub e-mail istnieje UserExistException