

Cezary Hołub  
Wrocław 2022

Wrocławska Wyższa Szkoła Informatyki Stosowanej

Przedmiot	Zaawansowane praktyki programistyczne
Semestr	Zima 2021/2022

Lab 8 05.02.2022

Materiały do ćwiczeń

**Uwaga!!! Lista wymaga Java 1.8 lub nowszej**  
**Lista nie jest na ocenę.**

## ZADANIE 1

Tworzenie projektu Mavenowego z zależnością do Spark Framework

### Cel ćwiczenia:

- Tworzenie projektu Mavenowego w Eclipse z zależnością do Spark Framework

### Przebieg ćwiczenia:

1. Proszę stworzyć w Eclipse projekt Mavenowy. File -> New -> Project -> [proszę znaleźć odpowiedni typ projektu]
2. Proszę wpisać jako Group Id: pl.wwsis.zpp, jako Artifact: MicroBlog
3. Eclipse sam stworzy plik pom.xml
4. W pliku pom.xml proszę dodać zależność do frameworka Spark 2.6 (zależność do biblioteki "jar"). Na stronie <http://www.sparkjava.com> proszę odnaleźć odpowiednią zależność mavenowa. Spark może wymagać innych zależności, proszę mieć to na uwadze.

```
<?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>spark-projekt</groupId>
  <artifactId>spark-projekt</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spark-projekt</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    [redacted]
  </dependencies>
</project>
```

5. W klasie którą Eclipse stworzył w metodzie main, proszę wpisać:

```
get("/hello", (req, res) -> "Hello World"); //proszę pamiętać o imporcie
statycznym
```

6. Proszę uruchomić program (używając Eclipseowego "Run As"). Proszę w przeglądarce wpisać

następujący URL: <http://localhost:4567/hello>

## ZADANIE 2

### Rozwiązanie błędu z wersją kompilatora Javy

#### Cel ćwiczenia:

- Ustawienie w pliku pom.xml odpowiedniej wersji kompilatora Javy, wymaganego przez wyrażenia regularne

#### Przebieg ćwiczenia:

1. Przy próbie kompilacji programu Mavenem dostaniemy błąd kompilacji typu

```
...lambda expressions are not supported in -source 1.5
(use -source 8 or higher to enable lambda expressions)...
```

Maven chce skompilować klasę z wyrażeniem lambda, z użyciem Javy 1.5. Wsparcie dla wyrażeń lambda pojawiło się w Javie 1.8.

2. Proszę rozwiązać błąd, są na to dwa sposoby (do wyboru)
  - a. w pliku pom.xml, poprzez dodanie dwóch wpisów w sekcji `<properties></properties>`
  - b. w pliku pom.xml, poprzez dodanie `maven-compiler-plugin` w sekcji `<plugins></plugins>`
3. Kompilacja projektu poprzez `mvn clean install` ma się zakończyć sukcesem.

## ZADANIE 3

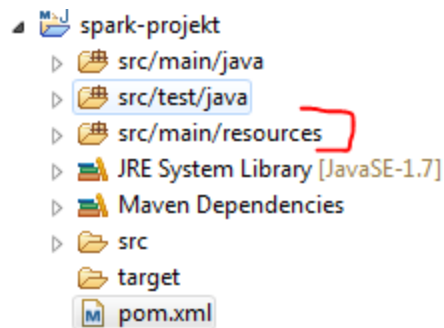
### Stworzenie prostego templaty z użyciem frameworka FreeMarker

#### Cel ćwiczenia:

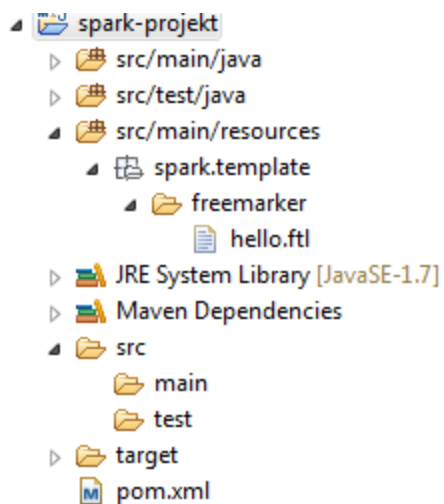
- Stworzenie prostego templaty z użyciem frameworka FreeMarker

#### Przebieg ćwiczenia:

1. Proszę znaleźć na stronie <http://www.sparkjava.com> zależność Mavenową do frameworka FreeMarker i uzupełnić plik pom.xml o tę zależność
2. Proszę stworzyć katalog `resources` w katalogu `src/main` a następnie nowo stworzony katalog dodać do BuildPath, aby uzyskać następującą strukturę:



3. Proszę stworzyć w katalogu `resources` ciąg katalogów `/spark/template/freemarker`. Ostatecznie w katalogu `freemarker` proszę stworzyć plik `hello.ftl`



4. Proszę w pliku `hello.ftl` dodać następującą treść:

```
<h1>${message}</h1>
<h1>Powyzszy tekst uzywa FreeMarkera</h1>
```

5. Proszę metodę `main` uzupełnić o następujący kod:

```
public static void main( String[] args )
{
    get("/hello", (request, response) -> {
        Map<String, Object> attributes = new HashMap<>();
        attributes.put("message", "Hello World!");

        return new ModelAndView(attributes, "hello.ftl");
    }, new FreeMarkerEngine());
}
```

6. Proszę program skompilować i uruchomić w przeglądarce.

#### ZADANIE 4

Dodanie własnych parametrów do obsługi przez wzorzec MVC

##### Cel ćwiczenia:

- Dodanie własnych parametrów do obsługi przez wzorzec MVC

##### Przebieg ćwiczenia:

1. Proszę dodać własne atrybuty do modelu (w metodzie `main`), które zostaną następnie przekazane do widoku. Proszę dodać wynik jakiejś operacji arytmetycznej lub inne dowolne parametry. Parametrów może być wiele.
2. Proszę dodać w szablonie `hello.ftl` odpowiedni parametr widoku FreeMarkera tak aby wydrukować wartość nowych atrybutów przekazanych przez kontroler.

#### ZADANIE 5

Obsługa różnych "routów" przez Sparka

##### Cel ćwiczenia:

- Dodanie nowych routów i odnotowanie jak na wywołanie każdej routy reaguje przeglądarka

##### Przebieg ćwiczenia:

1. Każde żądanie adresu URL w przeglądarce niesie ze sobą wywołanie odpowiedniego kontrolera obsługującego dane żądanie
2. Proszę stworzyć nową klasę i przepisać kod, następnie proszę wywołać w przeglądarce taki URL, który odniesie użytkownika od każdego z metody serwisowych po kolei.

```
package pl.wwsis.zpp.MicroBlog;

import static spark.Spark.*;

public class App {
    public static void main(String[] args) {

        get("/hello", (request, response) -> "Hello World");
    }
}
```

```

post("/hello", (request, response) -> {
    return request.body();
});

get("/private", (request, response) -> {
    response.status(401);
    return "go away!";
});

get("/users/:name", (request, response) -> {
    return "Selected user " + request.params(":name");
});

get("/news/:section", (request, response) -> {
    response.type("text/html");
    return "<?xml version='1.0' encoding='UTF-8'><news><b>" +
        request.params("section") + "</b></news>";
});

get("/protected", (request, response) -> {
    halt(403, "I don't think so!!!");
    return null;
});

get("/redirect", (request, response) -> {
    response.redirect("/news/world");
    return null;
});

get("/", (request, response) -> {
    return "root";
});
}
}

```

## ZADANIE 6

## Definicja routów dla metod serwisowych

Cel ćwiczenia:

- Definicja routów dla metod serwisowych

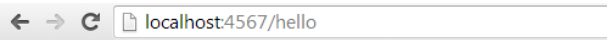
Przebieg ćwiczenia:

- Załóżmy że mamy jedną route w metodzie `main`:

```
public static void main( String[] args )
{
    get("/hello", (request, response) -> {
        Map<String, Object> attributes = new HashMap<>();
        attributes.put("message", "Hello World!");

        return new ModelAndView(attributes, "hello.ftl");
    }, new FreeMarkerEngine());
}
```

- Wywołanie w przeglądarce `localhost:4567/hello` oczywiście spowoduje pojawienie się strony



**Hello World!**

**Powyższy tekst używa FreeMarkera**

- Proszę tak zmodyfikować definicję route, aby każde odwołanie się w przeglądarce do `localhost:4567/{dowolny ciąg znaków}` wywołało naszą metodę serwisową i zwróciło stronę.

```
public static void main( String[] args )
{
    get("/{dowolny ciąg znaków}", (request, response) -> {
        Map<String, Object> attributes = new HashMap<>();
        attributes.put("message", "Hello World!");

        return new ModelAndView(attributes, "hello.ftl");
    }, new FreeMarkerEngine());
}
```

- Dla przykładu `localhost:4567/kkkkk` powinno zwrócić stronę:



**Hello World!**

**Pwyższy tekst używa FreeMarkera**