Warsztaty Spring MVC REST

v. 1.0.0

Cel warsztatu

Cel warsztatu jest napisanie funkcjonalności backendowej do katalogowania książek metodą REST.

W ramach warsztatu stworzymy API tożsame z tym które zostało udostępnione w ramach warsztatu poprzedniego (Javascript i Jquery).

Do stworzenie API wykorzystamy Spring MVC, dodatkową bibliotekę Jackson, oraz dodatkowe adnotacje.

Do ostatecznej weryfikacji poprawności naszego api wykorzystamy poprzedni frontendowy warsztat - zmieniając jedynie adres api, z którego ma ono korzystać.

Cel warsztatów

Server powinien mieś zaimplementowane ścieżki dostępu podane w tabeli:

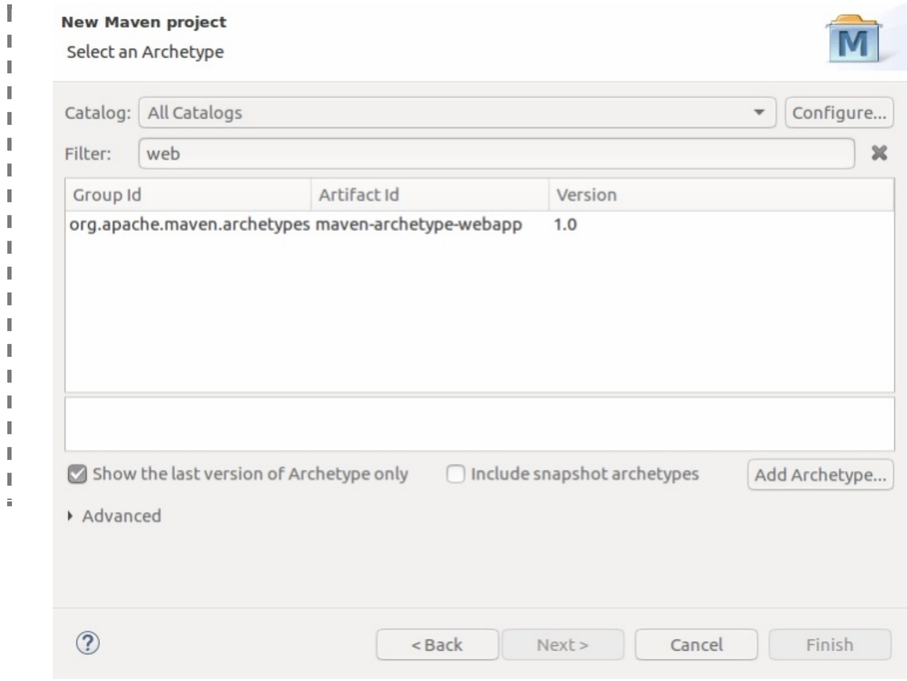
Metoda HTTP	ADRES	CO ROBI?
GET	/books/	Zwraca listę wszystkich książek.
POST	/books/	Tworzy nową książkę na podstawie danych przekazanych z formularza i zapisuje ją do bazy danych.
GET	/books/{id}	Wyświetla informacje o książce o podanym id.
PUT	/books/{id}	Zmienia informacje o książce o podanym id na nową.
DELETE	/books/{id}	Usuwa książkę o podanym id z bazy danych.

Przygotowanie

Zadanie 1 - tworzenie projektu

Ćwiczenia z wykładowcą

- Aby utworzyć projekt skorzystamy z artefaktu maven-archetype-webapp.
- Wybieramy go podczas tworzenia projektu Maven.



Tworzenie projektu

Definiujemy właściwości:

```
<properties>
<org.springframework-version>4.3.7.RELEASE
</org.springframework-version>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
<failonmissingwebxml>false</failonmissingwebxml>
</properties>
```

Uzupełniamy plik pom, dodając wymagane zależności.

```
<dependencies>
    <dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-webmvc</artifactid>
        <version>${org.springframework-version}</version>
    </dependency>
    <dependency>
        <groupid>javax.servlet</groupid>
        <artifactid>javax.servlet-api</artifactid>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Tworzenie projektu

Definiujemy właściwości:

```
<properties>
<org.springframework-version>4.3.7.RELEASE
</org.springframework-version>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
<failonmissingwebxml>false</failonmissingwebxml>
</properties>
```

Wersja Springa.
W chwili tworzenia prezentacji ta wersja jest najnowszą zalecaną.

Uzupełniamy plik pom, dodając wymagane zależności.

```
<dependencies>
    <dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-webmvc</artifactid>
        <version>${org.springframework-version}</version>
    </dependency>
    <dependency>
        <groupid>javax.servlet</groupid>
        <artifactid>javax.servlet-api</artifactid>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Tworzenie projektu

Definiujemy właściwości:

```
<properties>
<org.springframework-version>4.3.7.RELEASE
</org.springframework-version>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
<failonmissingwebxml>false</failonmissingwebxml>
</properties>
```

Wersja javy dla mavena.

Uzupełniamy plik pom, dodając wymagane zależności.

```
<dependencies>
    <dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-webmvc</artifactid>
        <version>${org.springframework-version}</version>
    </dependency>
    <dependency>
        <groupid>javax.servlet</groupid>
        <artifactid>javax.servlet-api</artifactid>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Tworzenie projektu

Definiujemy właściwości:

```
<properties>
<org.springframework-version>4.3.7.RELEASE
</org.springframework-version>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
<failonmissingwebxml>false</failonmissingwebxml>
</properties>
```

Informacja dla eclipse że w przypadku braku pliku web.xml ma nie zwracać błędu.

Uzupełniamy plik pom, dodając wymagane zależności.

```
<dependencies>
    <dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-webmvc</artifactid>
        <version>${org.springframework-version}</version>
    </dependency>
    <dependency>
        <groupid>javax.servlet</groupid>
        <artifactid>javax.servlet-api</artifactid>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

Tworzenie projektu

Definiujemy klasę konfiguracji.

Na tym etapie nasz projekt nie różni się od tworzonych wcześniej aplikacji z wykorzystaniem Spring MVC.

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "pl.coderslab")
public class AppConfig extends WebMvcConfigurerAdapter {


}
```

Tworzenie projektu

```
public class AppInitializer extends
  AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{AppConfig.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }
}
```

Dodajemy inicjalizator aplikacji.

Zwróć uwagę że korzystamy z innej nieco uproszczonej jego implementacji w porównaniu do tej z której korzystaliśmy wcześniej.

Zadania

Zadanie 1

Repozytorium

- Załóż nowe repozytorium Git na GitHubie..
- Pamiętaj o robieniu commitów (również co każde ćwiczenie).

- Stwórz plik .gitignore i dodaj do niego wszystkie podstawowe dane: (katalog z danymi twojego IDE, jeżeli istnieje, pliki *.class, itp.),

Zadanie 2

Książki

Utwórz model **Book**, który będzie przechowywał dane o książkach.

Jeżeli chcesz w jakiś sposób rozwinąć model, możesz to zrobić, pamiętaj tylko że będziemy testować api korzystając z poprzedniego warsztatu.

Klasa ma zawierać co najmniej:

- **id**: long,
- **isbn**: String,
- **title**: String,
- **author**: String,
- **publisher**: String,
- **type**: String,

Zadanie 3

Kontroller

Utwórz kontroller **BookController**, umieścimy w nim wszystkie metody wymagane przez nasze api.

Metoda testowa:

```
@RestController
@RequestMapping("/books")
public class BookController {

        @RequestMapping("/hello")
        public String hello(){
                return "{hello: World}";
        }
}
```

Zadanie 4 - Konwerter

W naszym projekcie moglibyśmy samodzielnie przekształcać obiekty na format JSON.

Nie jest to jednak zbyt wygodne.

Posłużymy się w tym celu biblioteką **Jackson**, należy uzupełnić zależności w pliku pom.xml:

```
<dependency>
    <groupid>com.fasterxml.jackson.core</groupid>
    <artifactid>jackson-databind</artifactid>
    <version>2.9.0.pr2</version>
</dependency>
```

Spring zadba o odpowiednie przekształcenia obiektu na JSON.

Przetestuj działanie tworząc i wywołując następującą akcję:

```
@RequestMapping("/helloBook")
public Book helloBook(){
        return new Book(1L,"9788324631766","Thiniking in Java",
                    "Bruce Eckel","Helion","programming");
}
```

Zadanie 5 - źródło danych

Utwórz klasę **MockBookList**, będzie to źródło danych dla naszej biblioteki.

Klasa ta powinna posiadać metody:

- Pobieranie listy dancyh.
- Pobieranie obiektu po wskazanym identyfikatorze.
- Edycje obiektu.
- Usuwanie obiektu.

Dodaj odpowiednią adnotację tak, aby klasa była komponentem zarządzanym przez Springa.

Utwórz zmienną tej klasy w kontrolerze, a następnie wstrzyknij ją.

Zadanie 5 - źródło danych

Prace rozpocznij od stworzenia listy książek oraz jej inicjalizacji.
Możesz wykorzystać poniższy przykład.

```
@Component
public class MockBookService {

	private List<Book> list;

	public MockBookService() {
		list = new ArrayList<>();
		list.add(new Book(1L, "9788324631766", "Thiniking in Java", "Bruce Eckel", "Helion", "programming"));
		list.add(new Book(2L, "9788324627738", "Rusz glowa Java.", "Sierra Kathy, Bates Bert", "Helion", "programming"));
		list.add(new Book(3L, "9780130819338", "Java 2. Podstawy", "Cay Horstmann, Gary Cornell", "Helion", "programming"));
	}

	public List<Book> getList() {
		return list;
	}

	public void setList(List<Book> list) {
		this.list = list;
	}


}
```

Zadanie 6

Punk dostępowy - lista wszystkich książek

Dodaj akcję kontrolera, która zwróci listę wszystkich książek, udostępnioną przez klasę **MockBookService.**

W przykładzie klasy **MockBookService** masz już odpowiednią metodę do pobrania listy.

Zadanie 7

Punk dostępowy - wybrana książka

Dodaj akcję kontrolera, na podstawie przekazanego parametru id zwróci wybraną książkę.

Uzupełnij klasę **MockBookService** o metodę która wyszuka w liście książkę o zadanym identyfikatorze .

Zadanie 7

Punk dostępowy - dodawanie książki

Dodaj akcję kontrolera, na podstawie przekazanego parametrów utworzy książkę a następnie doda ją do listy źródła danych.

Pamiętaj że przesyłanie danych w tym przypadku ma się odbywać za pomocą metody HTTP - POST.

Uzupełnij klasę **MockBookService** o metodę która doda książkę do listy .

Zadanie 8

Punk dostępowy - edycja książki

Dodaj akcję kontrolera, na podstawie przekazanego parametrów wyszuka a następnie zmodyfikuje książkę.

Pamiętaj że przesyłanie danych w tym przypadku ma się odbywać za pomocą metody HTTP - PUT.

Uzupełnij klasę **MockBookService** o metodę która modyfikuje książkę .

Zadanie 8

Punk dostępowy - uzunięcie książki

Dodaj akcję kontrolera, na podstawie przekazanego parametru wyszuka a następnie usunie książkę.

Pamiętaj że przesyłanie danych w tym przypadku ma się odbywać za pomocą metody HTTP - DELETE.

Uzupełnij klasę **MockBookService** o metodę która usunie książkę .

Zadanie 9

Testowanie api

Sprawdź poprawność metod GET za pomocą przeglądarki, dla metod POST, PUT, DELETE wykonaj poniższe polecenia przy uruchomionym serwerze:

Metoda **POST** - dodanie danych

```
curl -X POST -i -H "Content-Type: application/json" -d
  '{"isbn":"34321","title":"Thinking in Java",
  "publisher":"Helion","type":"programming",
  "author":"Bruce Eckel"}' http://localhost:8282/books/add
```

Metoda **PUT** - aktualizacja danych

```
curl -X PUT -i -H "Content-Type: application/json" -d
  '{"id":1,"isbn":"32222","title":"Thinking in Java",
  "publisher":"Helion","type":"programming",
  "author":"Bruce Eckel"}' http://localhost:8282/books/1/update
```

Metoda **DELETE** - usuwanie danych

```
curl -X DELETE -i  http://localhost:8282/books/remove/1
```

Po wykonaniu każdego polecenia sprawdzaj wyniki przy użyciu metody **GET**

Jeżeli kopiujesz polecenie - musi ono być w jednej lini.

Zadanie 9

Testowanie api

Zmodyfikuj aplikację frontendową utworzoną w ramach warsztatu **Javascript i jQuery: REST** tak by korzystała z utworzonego przez Ciebie punktu dostępowego.

Sprawdź poprawność działania aplikacji.