Warsztaty Spring MVC REST

v3.0

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:

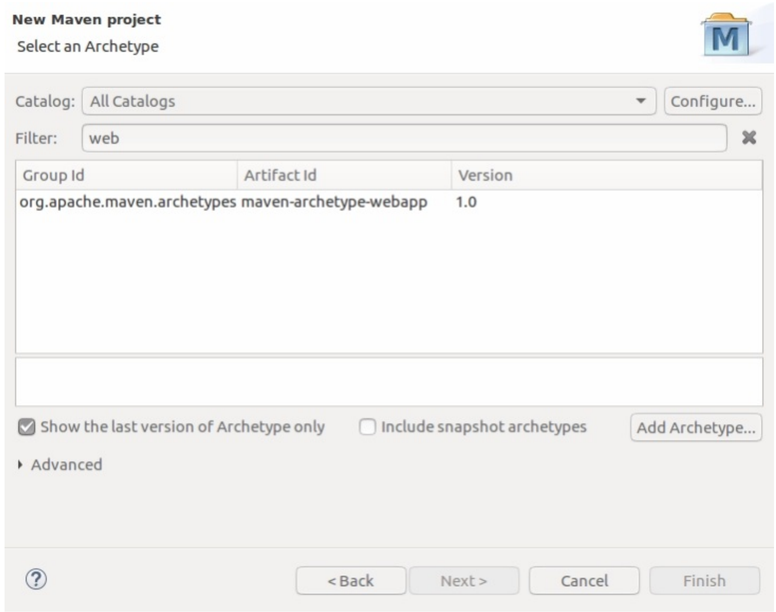
Metoda HTTP	ADRES	CO ROBI?
GET	/books/	Zwraca listę wszystkich książek.
POST	/books/	Tworzy nową książkę na podstawie danych przekazanych z formularza i zapisuje ją do bazy danych.
GET	/books/{id}	Wyświetla informacje o książce o podanym id.
PUT	/books/{id}	Zmienia informacje o książce o podanym id na 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>
  <failOnMissingWebXml>false</failOnMissingWebXml>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Tworzenie projektu

Definiujemy właściwości:

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

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

Tworzenie projektu

Definiujemy właściwości:

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

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

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

Tworzenie projektu

Definiujemy właściwości:

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

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

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

Wersja javy dla mavena.

Tworzenie projektu

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

Dodajemy inicjalizator aplikacji. Korzystamy z innej nieco uproszczonej jego implementacji w porównaniu do tej z której korzystaliśmy wcześniej.

```
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[]{"/"};   }
}
```

Tworzenie projektu

Dodajemy inicjalizator aplikacji. Korzystamy z innej nieco uproszczonej jego implementacji w porównaniu do tej z której korzystaliśmy wcześniej.

```
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[]{"/"};   }
}
```

Tworzenie projektu

Rejestrujemy filtr ustawiający odpowiednie kodowanie.

```
public class AppInitializer extends
  AbstractAnnotationConfigDispatcherServletInitializer {

    // ... kod z poprzedniego slajdu

    @Override
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter characterEncodingFilter =
                    new CharacterEncodingFilter();
        characterEncodingFilter.setEncoding("UTF-8");
        return new Filter[] { characterEncodingFilter };
    }
}
```

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 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}";
        }
    }
```

W odróżnieniu do MVC dodaliśmy adnotację **@RestController** - jest to połączenie znanych nam adnotacji: **@ResponseBody** i **@Controller**.

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>
```

Na podstawie wykrycia w projekcie biblioteki, Spring zadba o odpowiednie przekształcenia obiektu na JSON.

Zadanie 4 - Konwerter

Przetestuj działanie, tworząc i wywołując następującą akcję:

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

Zadanie 5 - źródło danych

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

Klasa ta powinna posiadać metody:

- Pobieranie listy danych.
- 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 MemoryBookService {
    private List<Book> list;
    public MemoryBookService() {
        list = new ArrayList<>();
        list.add(new Book(1L, "9788324631766", "Thinking 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 **MemoryBookService.**

W przykładzie klasy **MemoryBookService** 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 **MemoryBookService** 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 **MemoryBookService** 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 **MemoryBookService** o metodę, która modyfikuje książkę.

Zadanie 8

Punk dostępowy - usunięcie 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 **MemoryBookService** o metodę która usunie książkę .

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.

Zadanie 10

Utwórz interfejs **BookService**, zawierający wszystkie metody **MemoryBookService**.

Zmodyfikuj klasę **MemoryBookService** tak by implementowała utworzony interfejs.

Zmodyfikuj kontroler tak by wstrzykiwać obiekt typu **BookService**.

Snippety - @RequestBody

Wykorzystaj adnotację **@RequestBody** - dzięki niej **Spring** wyszuka a następnie skorzysta z odpowiedniego konwertera by przekształcić dane **HTTP** na obiekt Javy.

Przykład:

```
@PostMapping("/books")
public void addBook( @RequestBody Book book){
    // operacje na obiekcie book
}
```

Zapoznaj się z dokumentacją

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-requestbody

Snippety - CORS

Na stronie można wykonywać żądania **AJAX** tylko w obrębie tej samej **origin** co oznacza, ten sam:

- protokół
- host
- port

Jest to związane z zasadą **Same-origin policy** - więcej na ten temat: https://en.wikipedia.org/wiki/Same-origin_policy

Snippety - CORS

Jeżeli ta zasada nie jest zachowana otrzymamy w konsoli przeglądarki komunikat analogiczny do poniższego:

```
XMLHttpRequest cannot load http://localhost:8080/Warsztaty5/books/all.
No 'Access-Control-Allow-Origin' header is present on the requested resource.
Origin 'null' is therefore not allowed access.XMLHttpRequest cannot
                      load http://localhost:8080/Warsztaty5/books/all.
No 'Access-Control-Allow-Origin' header is present on the requested resource.
Origin 'null' is therefore not allowed access.
```

Snippety - CORS

Rozwiązaniem tego problemu, jest odpowiednia konfiguracja naszej aplikacji, możemy ją globalnie dodać w pliku konfiguracji nadpisując metodę **`addCorsMappings`**

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
        .allowedMethods("GET", "POST", "PUT", "DELETE")
        .allowedOrigins("http://localhost");
    }
}
```

Snippety - CORS

Rozwiązaniem tego problemu, jest odpowiednia konfiguracja naszej aplikacji, możemy ją globalnie dodać w pliku konfiguracji nadpisując metodę **addCorsMappings**

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
        .allowedMethods("GET", "POST", "PUT", "DELETE")
        .allowedOrigins("http://localhost");
    }
}
```

Określamy adres dla jakiego obsługa **CORS** ma być włączona, jeżeli api naszej aplikacji jest dostępne pod adresem **/api** określamy **"/api/**"**.

Snippety - CORS

Rozwiązaniem tego problemu, jest odpowiednia konfiguracja naszej aplikacji, możemy ją globalnie dodać w pliku konfiguracji nadpisując metodę **`addCorsMappings`**

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
        .allowedMethods("GET", "POST", "PUT", "DELETE")
        .allowedOrigins("http://localhost");
    }
}
```

Określamy metody dla jakich obsługa ma być włączona.

Snippety - CORS

Rozwiązaniem tego problemu, jest odpowiednia konfiguracja naszej aplikacji, możemy ją globalnie dodać w pliku konfiguracji nadpisując metodę **addCorsMappings**

```
@Configuration
@EnableWebMvc
public class WebConfig extends WebMvcConfigurerAdapter {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
        .allowedMethods("GET", "POST", "PUT", "DELETE")
        .allowedOrigins("http://localhost");
    }
}
```

Określamy adres z jakiego dozwolony będzie dostęp. W przypadku wywołania pliku html bezpośrednio z systemu plików za pomocą przeglądarki możemy tą linię zakomentować.

Snippety - CORS

Więcej na ten temat:

https://docs.spring.io/spring/docs/current/spring-framework-reference/html/cors.html

https://spring.io/blog/2015/06/08/cors-support-in-spring-framework

https://spring.io/guides/gs/rest-service-cors/

https://zinoui.com/blog/cross-domain-ajax-request

https://enable-cors.org/

Możliwe rozszerzenia

Utwórz bazę danych z tabelą dla książek oraz implementację klasy **BookDao** zawierającą metody do jej obsługi.

Utwórz serwis **DbBookService** implementującą **BookService**, która będzie korzystać z **BookDao**.

Zmodyfikuj kontroler, tak by wstrzykiwał obiekt typu **BookService**.

Za pomocą znanych mechanizmów określ która implementacja ma być wstrzykiwana.

Możliwe rozszerzenia

Dodaj klasę **Author** oraz odpowiadającą mu klasę **AuthorDao**.

Utwórz klasę serwisu do obsługi autorów.

Dodaj kontroler dla klasy **Author** udostępniający metody analogiczne jak w przypadku kontrolera dla klasy **Book**.

Dodaj relację pomiędzy autorami a książkami.