RESTFul Web Services with Spring

Lab Book

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision No. | Author | **Summary of Changes** |
| May 2016 | 1.0 | Yukti A Valecha | Created new lab book as per revised course contents |
|  |  |  |  |
| April 2018 | 2.0 | Anjulata | Modified the lab book as per revised course contents |
|  |  |  |  |

Table of Contents

[Getting Started 4](#_Toc510986659)

[Overview 4](#_Toc510986660)

[Setup Checklist for Web Services 2.0 4](#_Toc510986661)

[Instructions 4](#_Toc510986662)

[Lab 1. Web Services (JAX-WS) 5](#_Toc510986663)

[Lab 2. Web Services (JAX-RS) and Spring REST 6](#_Toc510986664)

[Lab 3. Spring REST: Exception Handling, Versioning and Pagination 8](#_Toc510986665)

[Appendices 9](#_Toc510986666)

[Appendix A: Naming Conventions 9](#_Toc510986667)

[Appendix B: Table of Figures 10](#_Toc510986668)

Getting Started

## Overview

This lab book is a guided tour for learning Introduction to Web Services (SOAP and REST) version 2.0. It comprises ‘To Do’ assignments.

## Setup Checklist for Web Services 2.0

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

Intel Pentium 90 or higher (P166 recommended)

Microsoft Windows 95, 98, or NT 4.0, 2k, XP.

Memory: 32MB of RAM (64MB or more recommended)

Internet Explorer 9.0 or higher

JDK1.8 or above

Wildfly Server 8.0 or above.

Spring Tool Suit 3.9 or higher

## Instructions

For all naming conventions refer Appendix A. All lab assignments should refer coding standards.

Create a directory by your name in drive <drive>. In this directory, create a subdirectory webservices\_assgn. For each lab exercise create a directory as lab <lab number>.

You may also look up the on-line help if necessary.

1. Web Services (JAX-WS)

|  |  |
| --- | --- |
| Goals | Understand the process of creating and Consuming a Java Web Service |
| Time | 50 minutes |

1. Refer below Java files:

* [Product](Product.java).java
* [ProductDB](ProductDB.java).java

Create a Product Web service that will accept the product name and return the price of the product to the web service consumer.

If the product is not available then the web service should return a message to the consumer specifying that the product is not available.

**Note:** Make use of the static DB given in ProductDB.java

Refer below screen shots:

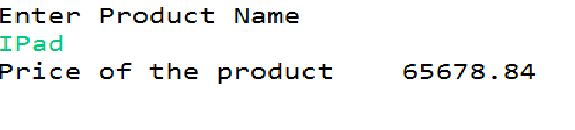


Figure : Screenshot

If the user enters a product that does not exists then, refer below screen shot

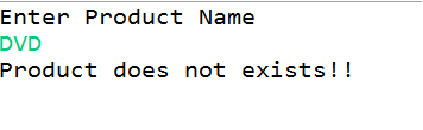


Figure : Screenshot

1. Web Services (JAX-RS) and Spring REST

|  |  |
| --- | --- |
| Goals | Understand the process of creating and Consuming a RESTful Java Web Service |
| Time | 1. minutes |

1. Refer below Java files:

<Product.java>

<ProductDB.java>

Create a Product RESTful web service that will display all products to the Web service consumer.

**Note:** Make use of the static DB given in ProductDB.java

Refer below screen shot:

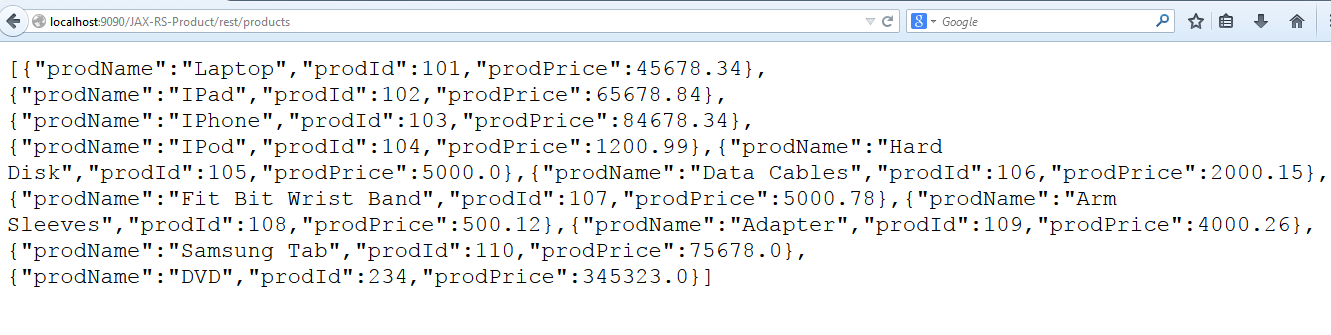


Figure : Screenshot

The consumer should also be able to add a product to the existing list of products.

Refer below screen shots:

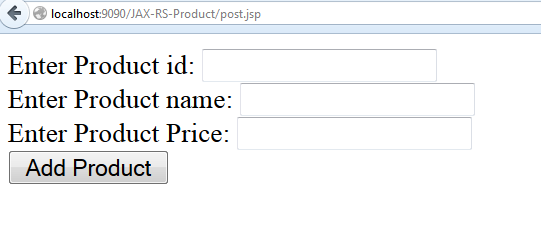


Figure : Input Screen

After entering product details, refer below screen shot

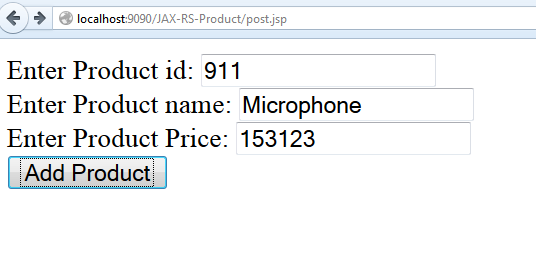


Figure : Input Screen with data

Product is added to existing list of products

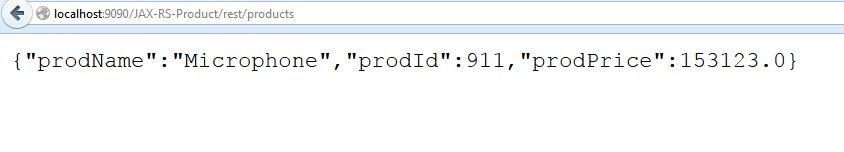


Figure : Screen shot

Now, new product is added to existing list of products

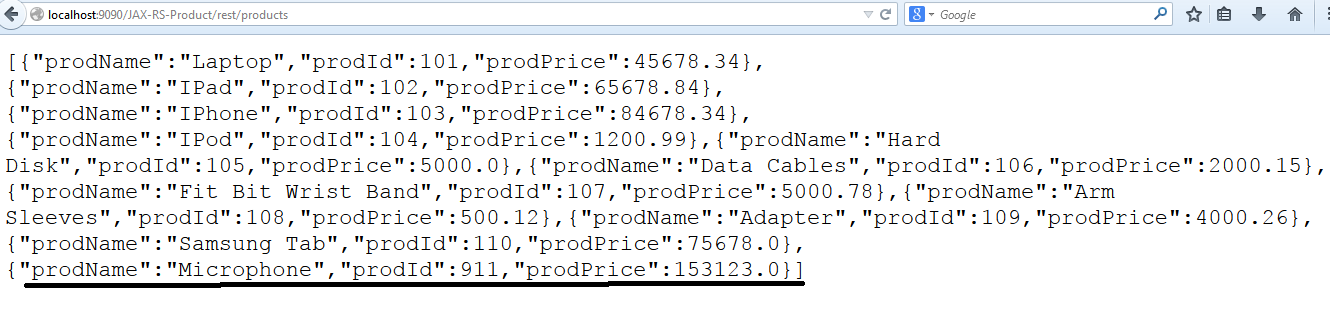


Figure : Screen shot

1. Refer to the above assignment and create with Spring REST.
2. Spring REST: Exception Handling, Versioning and Pagination

|  |  |
| --- | --- |
| Goals | Understand the process of creating and Consuming a RESTful Java Web Service |
| Time | 1. minutes |

1. Refer to the Spring REST application created in Lab 2.2. Apply the Exception handling.
2. In Previous lab add the category property in the product class and add the versioning features as new product data will be available with V1 link.

[Hint: Use URI Versioning]

Appendices

## Appendix A: Naming Conventions

**Package** names are written in all lower case to avoid conflict with the names of classes or interfaces. Companies use their reversed Internet domain name to begin their package names—for example, com.cg.learning.mypackage for a package named learning.mypackage created by a programmer at cg.com.

Packages in the Java language itself begin with **java. Or javax**.

**Classes and interfaces** the first letter should be capitalized, and if several words are linked together to form the name, the first letter of the inner words should be uppercase (a format that's sometimes called "camelCase").

For classes, the names should typically be nouns. For example:

***Dog***

***Account***

***PrintWriter***

For interfaces, the names should typically be adjectives like

***Runnable***

***Serializable***

**Methods** The first letter should be lowercase, and then normal camelCase rules should be used. In addition, the names should typically be verb-noun pairs. For example:

***getBalance***

***doCalculation***

***setCustomerName***

**Variables** Like methods, the camelCase format should be used, starting with a lowercase letter. Sun recommends short, meaningful names, which sounds good to us. Some examples:

***buttonWidth***

***accountBalance***

***myString***

**Constants** Java constants are created by marking variables static and final. They should be named using uppercase letters with underscore characters as separators:

***MIN\_HEIGHT***

## Appendix B: Table of Figures

[Figure 1 : Screenshot 4](#_Toc451263958)

[Figure 2 : Screenshot 4](#_Toc451263959)

[Figure 3 : Screenshot 5](#_Toc451263960)

[Figure 4 : Input Screen 5](#_Toc451263961)

[Figure 5 : Input Screen with data 6](#_Toc451263962)

[Figure 6 : Screen shot 6](#_Toc451263963)

[Figure 7 : Screen shot 6](#_Toc451263964)