EECS 4413. LAB 06: Data persist: JDBC. DAO design pattern

A. IMPORTANT REMINDERS

- **This lab will be graded.**
- Lab6 is due on **Saturday (May 18) at 11pm**. No late submission will be accepted.
- For this lab, you are welcome to attend the virtual lab sessions on Saturdays. ~~TAs or instructor will be available to help you. The location is LAS1002. Attendance is optional.~~ You can also ask questions after class.
- Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too.
- You can submit your lab work any time before the specified deadline.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

- Download this lab description and the associated files and read it completely.

C. GOALS/OUTCOMES FOR LAB

- On JDBC, both in MySQL, and SQLite.
- DAO design pattern
- Filters
- Restful web services

D. TASKS

Part1A: JDBC connection -- MYSQL

Part1B: DAO and MVC model

Part 2: JDBC connection -- SQLite

Part 3: Filters (not graded)

Part 4: RESTful web services

E. SUBMISSIONS

- eClass submission. More information can be found at the end of this document.

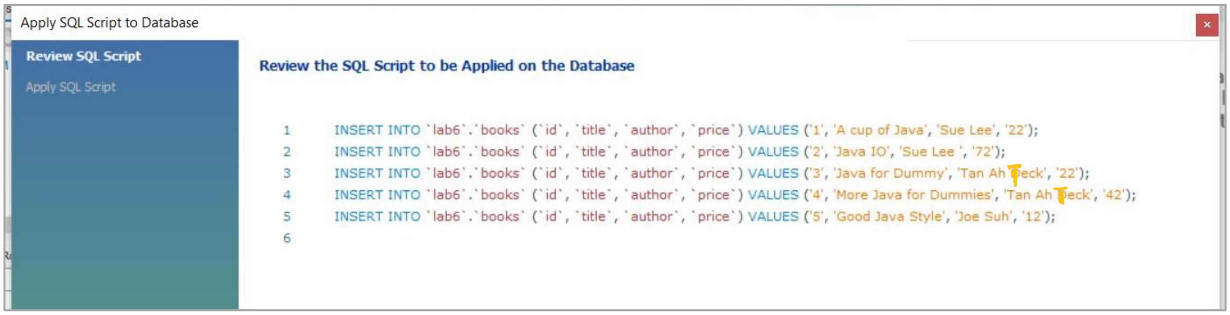
Part I JDBC, DAO model

In this exercise, we extend the bookshop program you did in lab3~5, but with the book data in a database. In this exercise we use MySQL database. In Part 2 we use another database SQLite.
In the previous versions, books are hard-coded and maintained in class Table. Now, we put the data in database.

Part 1A

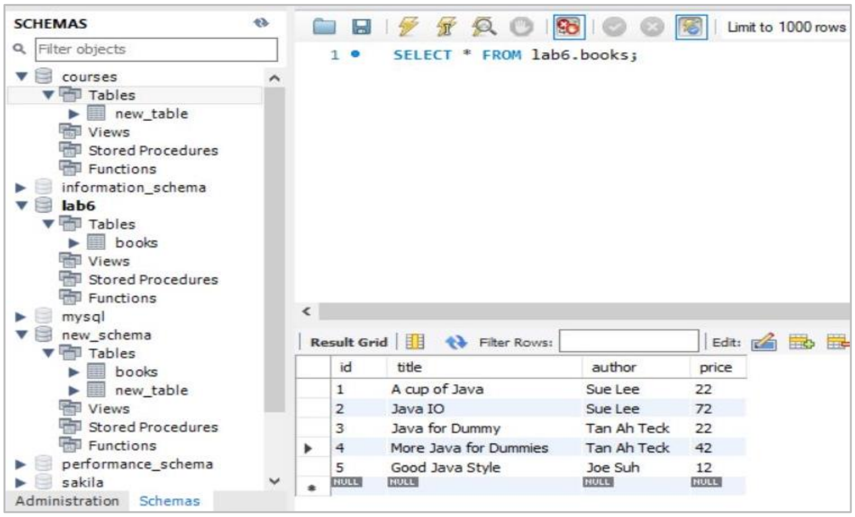
- Download MySQL Server (Community version) from https://dev.mysql.com/downloads/ and install on your computer. You can install it as Service or as an Application. During installation or configuration, when asked, use **EECS4413** as the password for **root** user. This allows the TA to run your program without modifying the password.
- You can manipulate MySQL with command line, using the shell it provides or your terminal, but it is easier to use a GUI tool. Download a GUI for MySQL. One popular one is **MySQL WorkBench**. If the installation already has the WorkBench GUI tool, you can use it. Otherwise, you can download MySQL WorkBench separately or download other GUI tools that can connect to MySQL (e.g., Heidi SQL).
- To operate on the database from Java, you should also have a jar file (the database driver). If your installation already has a folder Connector J, then you can find the jar file there. Otherwise download the Connector J separately for your version of OS and MySQL. Two recent versions of the connector jar files are also provided for you.
- Start the MySQL server, and use command line or a GUI tool such as MySQL WorkBench to create a Schema/Database **lab6** and then create a table **books** in the schema/database, with the structure and data as follows:

```
CREATE TABLE `books` (
 `id` int NOT NULL,
 `title` varchar(45) DEFAULT NULL,
 `author` varchar(45) DEFAULT NULL,
 `price` double DEFAULT NULL,
 PRIMARY KEY (`id`)
```



Review the SQL Script to be Applied on the Database

```
INSERT INTO `lab6`.`books` (`id`, `title`, `author`, `price`) VALUES ('1', 'A cup of Java', 'Sue Lee', '22');
INSERT INTO `lab6`.`books` (`id`, `title`, `author`, `price`) VALUES ('2', 'Java IO', 'Sue Lee ', '72');
INSERT INTO `lab6`.`books` (`id`, `title`, `author`, `price`) VALUES ('3', 'Java for Dummy', 'Tan Ah Teck', '22');
INSERT INTO `lab6`.`books` (`id`, `title`, `author`, `price`) VALUES ('4', 'More Java for Dummies', 'Tan Ah Teck', '42');
INSERT INTO `lab6`.`books` (`id`, `title`, `author`, `price`) VALUES ('5', 'Good Java Style', 'Joe Suh', '12');
```



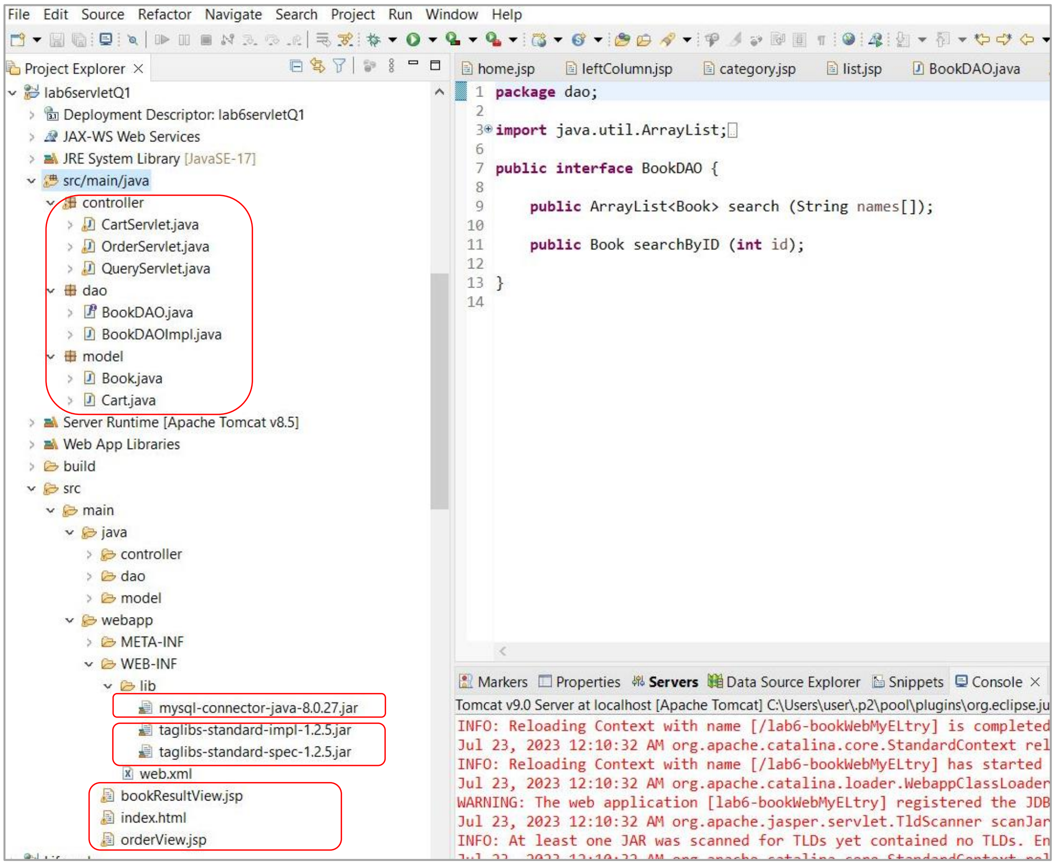
```
SELECT * FROM lab6.books;
```

	id	title	author	price
	1	A cup of Java	Sue Lee	22
	2	Java IO	Sue Lee	72
	3	Java for Dummy	Tan Ah Teck	22
▶	4	More Java for Dummies	Tan Ah Teck	42
	5	Good Java Style	Joe Suh	12
*	NULL	NULL	NULL	NULL

- Now import the **lab5servletQ2B.war** you submitted for lab5, import as **lab6servletQ1**.
- Remove the class Table.
- Modify the code of **QueryServlet** so that it given author names, loads the list of books from the database.
- If you have implemented the search for book in **CartServlet**, then modify the code of it so that it loads from the database.
- Make sure MySQL server is running. Put the connector jar file into the lib directory of your project, as demonstrated in class and the video.
- Run the program. If implemented correctly, your program should give the same output as in lab5.

Part 1B DAO design pattern (as well as MVC model)

- The drawback of part 1 is that the database connect code is intermixed with the business logic code. It is good to have clear separation of business logic and low level database connection and retrieval. Moreover, if database connection information needs to be changed (e.g., using another database jar file), then all the connection code needs to be modified. A cleaner approach is the DAO pattern where database connection is handled by a dedicated class.
- If you haven't done in part 1A, create a package **controller**, put the servlet files into the package. Create **model** package, put Book and Cart class into there.
- Create a new package **dao**. Inside, create an interface **BookDAO**, which defines methods **public ArrayList<Book> search (String authors[])** and **public Book searchByID(int id)**
- In the same package, create a class **BookDAOImpl**, which implements the interface, handling all database connection and retrieve books by authors or id. The structure of the project is given in the figure below.
- In **QueryServlet**, remove the database connection and data retrieval code that you did in part 1A. Instead, to get a list of books, create a instance of **BookBAOImpl** and call method on it (similar to call **Table** in previous labs). Following the principle of 'programming to interface', assign the instance to type **BookDAO**
- Make a similar change if you do book search in **CartServlet**.
- If implemented correctly, your program should get the same output.



```
lab6servletQ1
  Deployment Descriptor: lab6servletQ1
  JAX-WS Web Services
  JRE System Library [JavaSE-17]
  src/main/java
    controller
      CartServlet.java
      OrderServlet.java
      QueryServlet.java
    dao
      BookDAO.java
      BookDAOImpl.java
    model
      Book.java
      Cart.java
  Server Runtime [Apache Tomcat v8.5]
  Web App Libraries
  build
  src
    main
      java
        controller
        dao
        model
      webapp
        META-INF
        WEB-INF
          lib
            mysql-connector-java-8.0.27.jar
            taglibs-standard-impl-1.2.5.jar
            taglibs-standard-spec-1.2.5.jar
          web.xml
        bookResultView.jsp
        index.html
        orderView.jsp
```

```java
package dao;

import java.util.ArrayList;

public interface BookDAO {

    public ArrayList<Book> search (String names[]);

    public Book searchByID (int id);

}
```

```
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Users\user\.p2\pool\plugins\org.eclipse.ju
INFO: Reloading Context with name [/lab6-bookWebMyELtry] is completed
Jul 23, 2023 12:10:32 AM org.apache.catalina.core.StandardContext rel
INFO: Reloading Context with name [/lab6-bookWebMyELtry] has started
Jul 23, 2023 12:10:32 AM org.apache.catalina.loader.WebappClassLoader
WARNING: The web application [lab6-bookWebMyELtry] registered the JDB
Jul 23, 2023 12:10:32 AM org.apache.jasper.servlet.TldScanner scanJar
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. En
```

- Submit: export as was file with default name **lab6servletQ1.war**. Make sure to check 'export source files'

Part 2 JDBC, SQLite

In the exercise, you implement a Book search application, using SQLite, with more tables.

This application maintains a database of books. It allows users to search all books, search book by category, and can search by keyword.
In this application we use another database system: SQLite. SQLite is a serverless embedded database. Database is stored in a .db disk file. You access the database by accessing the .db file.

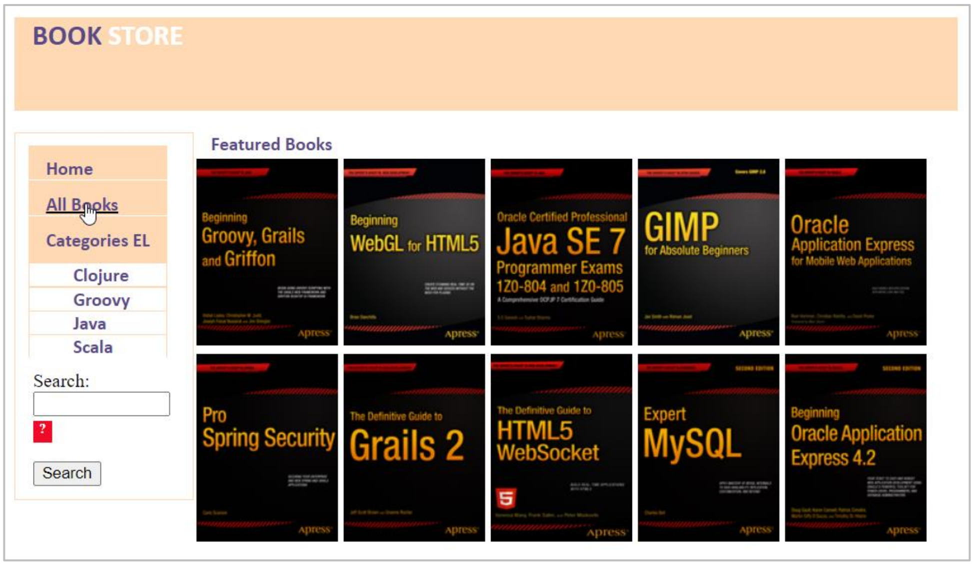
- You are given **Books.db**, which is the SQLite database file used by the application. It contains 3 tables. Download the file to your computer. You can use command line to access manipulate the database, but you can download a GUI for it. Popular ones are **DB browser for SQLite** and **SQLite Studio.** With this tool you can open the database and view the tables there. The 3 tables are: BOOK, AUTHOR, CATEGORY. Study the structure of the tables.

- You are given the partially implemented program. Import the provided war file, as **lab6serveltQ2.** The program implements MVC and DAO pattern. The program contains Servlet, JSP, CSS and JS files. Study how the different languages implement different functionalities, which may help you for your project.
- Read the code and complete the code.
  - Complete the DAO class **BookDAOImpl**. Study the database and you will need to join 3 tables to get the needed information from the database. You need to point to the path of the database file **Books.db**.
  - Complete the controller file **BookController**, which uses DAO to retrieve the books.
  - Complete JS file **list.jsp** to output list of books. Use JSTL and EL. Don't use scriptlets to embed java code

- Put the provided SQLite JDBC jar file to the lib directory of your project.
- If implemented correctly, you will get the following results.

Home page.

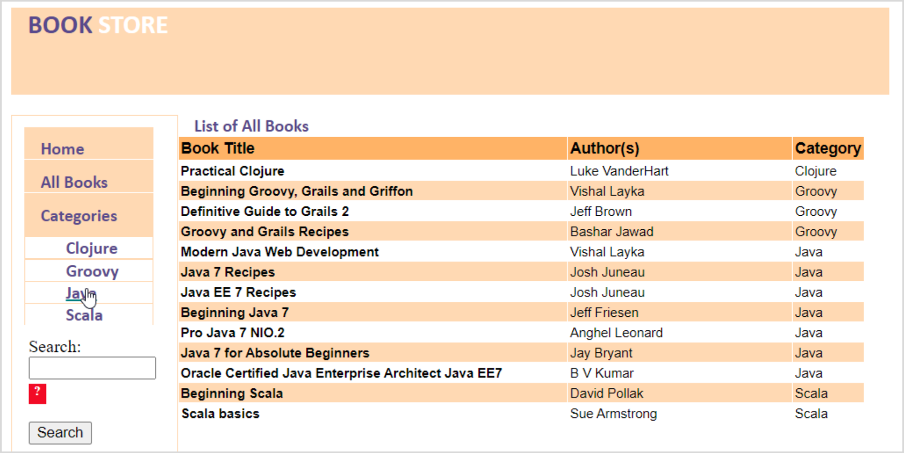
Then click All Books


BOOK STORE
Home
All Books
Categories EL
Clojure
Groovy
Java
Scala
Search:
?
Search
Beginning Groovy, Grails and Griffon
Beginning WebGL for HTML5
Oracle Certified Professional Java SE 7 Programmer Exams 1Z0-804 and 1Z0-805
GIMP for Absolute Beginners
Oracle Application Express for Mobile Web Applications
Pro Spring Security
The Definitive Guide to Grails 2
The Definitive Guide to HTML5 WebSocket
Expert MySQL
Beginning Oracle Application Express 4.2

Featured Books

All Books search result. Totally 13 books.

Then, click Java Category

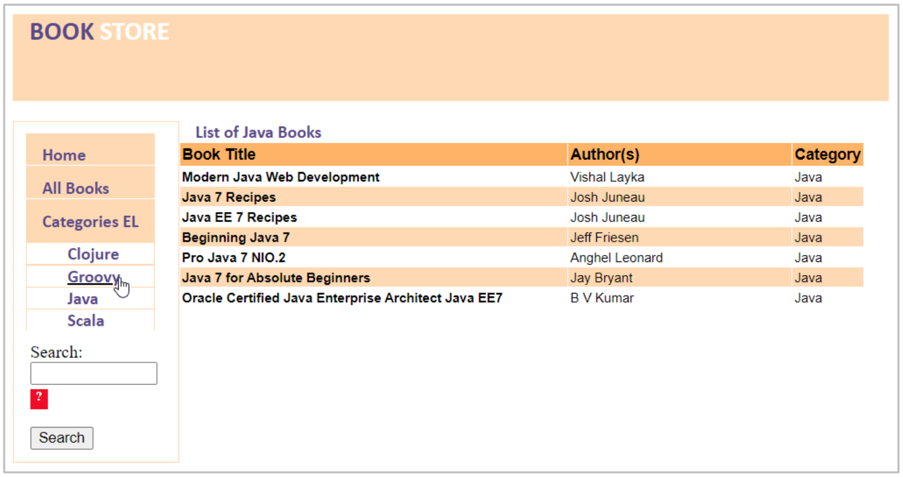


List of All Books

Book Title	Author(s)	Category
Practical Clojure	Luke VanderHart	Clojure
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Definitive Guide to Grails 2	Jeff Brown	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy
Modern Java Web Development	Vishal Layka	Java
Java 7 Recipes	Josh Juneau	Java
Java EE 7 Recipes	Josh Juneau	Java
Beginning Java 7	Jeff Friesen	Java
Pro Java 7 NIO.2	Anghel Leonard	Java
Java 7 for Absolute Beginners	Jay Bryant	Java
Oracle Certified Java Enterprise Architect Java EE7	B V Kumar	Java
Beginning Scala	David Pollak	Scala
Scala basics	Sue Armstrong	Scala

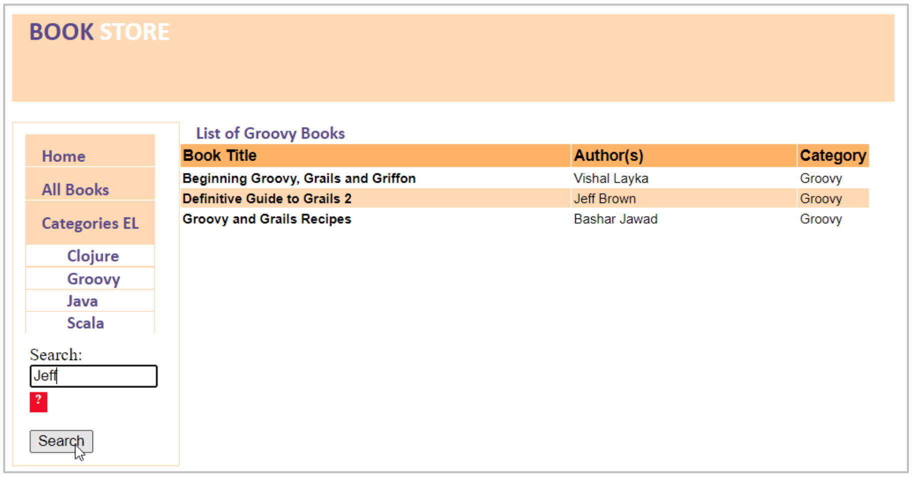
Java category search result.

Click Groovy Category



List of Java Books

Book Title	Author(s)	Category
Modern Java Web Development	Vishal Layka	Java
Java 7 Recipes	Josh Juneau	Java
Java EE 7 Recipes	Josh Juneau	Java
Beginning Java 7	Jeff Friesen	Java
Pro Java 7 NIO.2	Anghel Leonard	Java
Java 7 for Absolute Beginners	Jay Bryant	Java
Oracle Certified Java Enterprise Architect Java EE7	B V Kumar	Java



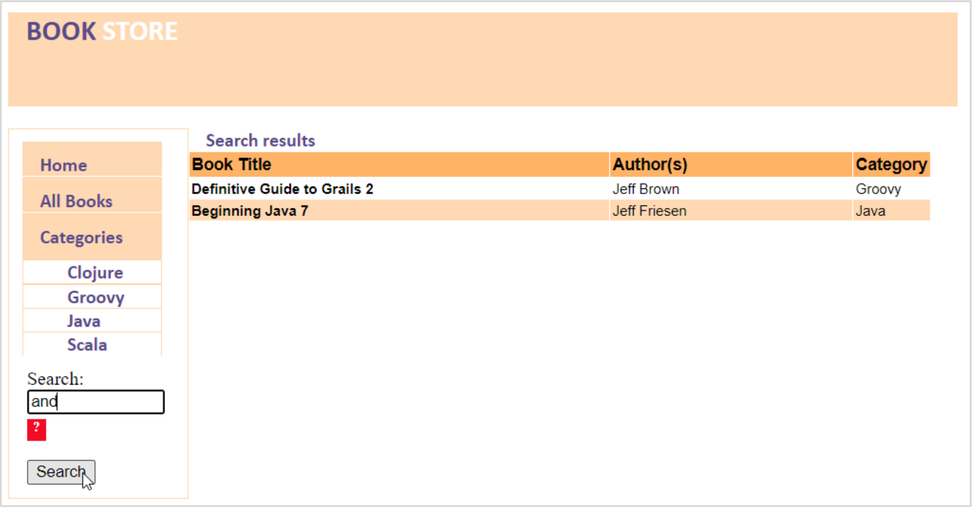
List of Groovy Books

Book Title	Author(s)	Category
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Definitive Guide to Grails 2	Jeff Brown	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy

Search: Jeff

Groovy category search result.

Then, search by keyword 'Jeff'



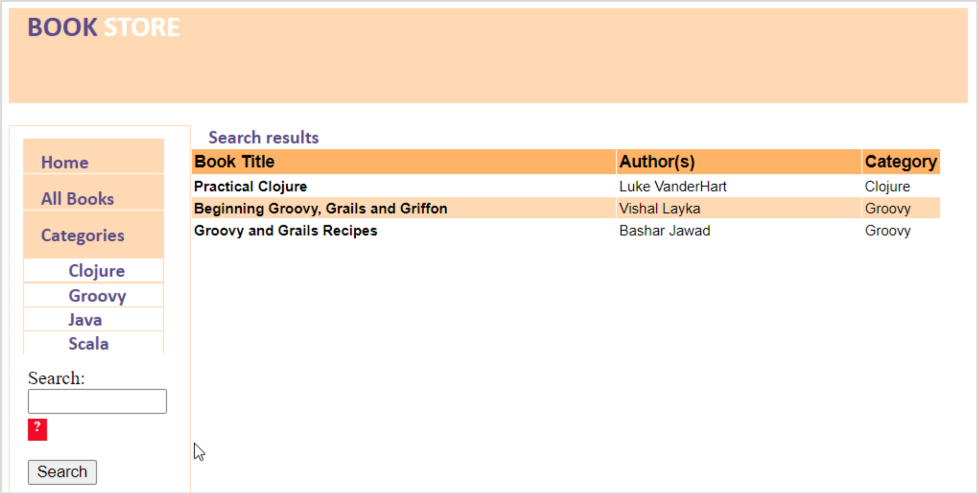
Search results

Book Title	Author(s)	Category
Definitive Guide to Grails 2	Jeff Brown	Groovy
Beginning Java 7	Jeff Friesen	Java

Search: and

Keyword 'Jeff' search result

Then, search by keyword 'and'



Search results

Book Title	Author(s)	Category
Practical Clojure	Luke VanderHart	Clojure
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy

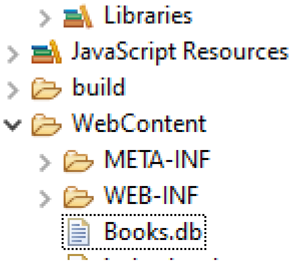
Keyword 'and' search result

See video for more information.

- Submission:
  So far, the database file Books.db is stored externally to the project. When you give the project to other users, you need to provide both the application war file and the database file. Moreover, users need to modify the Java

code for the database file's absolute path when they download the database file into their computers. Sometime it may be more convenient to store the database file into the project folder so it is included in the war file. Since the file path needs to be absolute, you need to use some method to convert from relative path in the project to the absolute path on the computer.

- In eclipse, copy the database file into the **WebContent** folder of your project.



- Change the absolute path into this:

```
String path = this.getServletContext().getRealPath("/Books.db") ;
Connection con=DriverManager.getConnection("jdbc:sqlite:" + path);
```

Note: your DAO class may not have servletContext. You can somehow pass servletContext object from Servlet class to this class.

Test it and make sure it works.

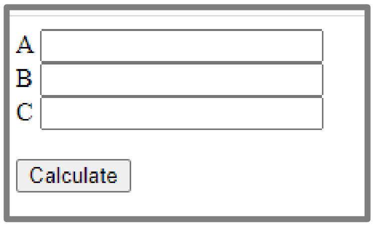
You can also create a folder e.g., **dbFile** under **WebContent**. Then the relative path will be "**/dbFile/Books.db**"

- export as war file with default name **lab6servletQ2.war**. Make sure to check 'explort source files'
  - you may want to transfer the war file to another computer, and run it, to make sure the database file is accessible without any code change.

Hope this program help you with the product search part of the project.

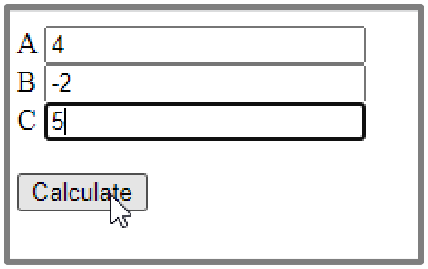
Part III filter (no submission, not graded)

Refactor the program you did for labtest1, which takes 3 numbers A, B, C that represent the lengths of all three semi-axes of an ellipsoid, and submit to a servlet to calculate the volume of the ellipsoid, using the formula Volume = 4/3 × π × *A* × *B* × *C*.



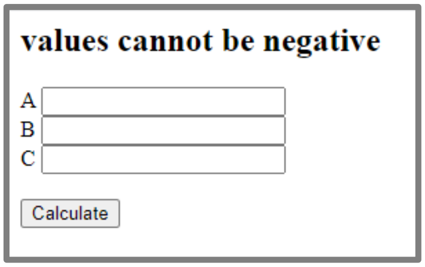
The first servlet receives the inputs and calculates the volume, and then forward the result to another servlet or JSP to display the result.
Note that unlike in labtest, now the servlet that handles the inputs should never check the values – just calculate with the formula. So remove the code of negative checking.

- Add a filter to the program such that if one or more of the 3 values are negative, the program displays "**values cannot be negative**" and loads the form again. Note that in this case the servlet or JSP is never executed.
- If no values are negative, then the filter pass the data to the servlet or JSP to calculate.



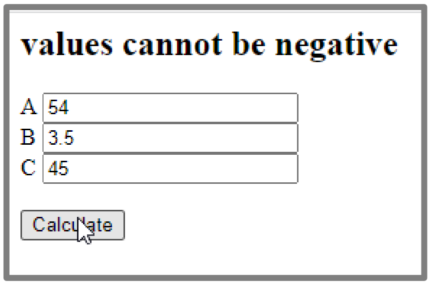
A 4
B -2
C 5

⇨



values cannot be negative

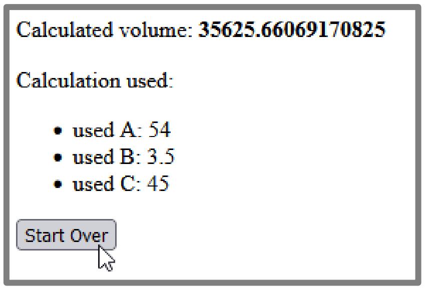
A
B
C



values cannot be negative

A 54
B 3.5
C 45

⇨



Calculated volume: **35625.66069170825**

Calculation used:

- used A: 54
- used B: 3.5
- used C: 45

Part IV Web services with REST API

In this part you get exposure to the RESTful web services. Particularly JAX-RS API and its reference implementation Jersey.

Exercise1 (no submission, not graded): create a JAX-RS RESTful web service from scratch.
Create a new dynamic project, name it, say, *tryRESTjar*.
Copy the provided Jersey jar files to your project lib folder.
To use Jersey, modify the *web.xml* by adding the servlet dispatcher, as shown in slides and below.

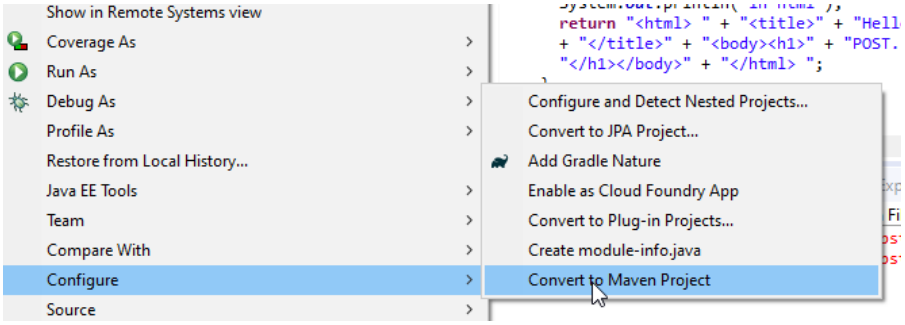
```
 <servlet>
    <servlet-name>your servlet name</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>your java package name</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>     <!-- this is optional -->
  </servlet>
  <servlet-mapping>
    <servlet-name>your servlet name</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
```

Create a java package, and a class (not servlet) in it, copy the provided code of *Hello.java* there.
Understand how different annotations (@path, @GET, @Produces).
Run the program on server. Use browser or curl to connect to the service. To request particular message format, set Accept header field, using *-H Accept:* in curl.

curl localhost:8080/tryRESTjar/rest/hello -H "Accept: text/xml"
curl localhost:8080/tryRESTjar/rest/hello -H "Accept: text/html"
curl localhost:8080/tryRESTjar/rest/hello -H "Accept: applicaton/json"
curl localhost:8080/tryRESTjar/rest/hello -X POST

Exercise2: **(no submission, not graded)** create a JAX-RX RESTful web service from Maven.
In previous exercise we add jar files manually. An easier, more common approach is to use Maven to download the dependencies for you.

Create a new dynamic project. Modify the *web.xml* by adding the servlet dispatcher, as before.
Create a java class (not servlet), copy the provided code there.
New, go to configure, select ‘convert to Maven project’. This will convert the project to a Maven project, which has a *pom.xml* file that specifies the dependencies. (You can create a Maven project directly but converting from a dynamic project is a bit easier at this stage)



```
return "<html> " + "<title>" + "Hell
+ "</title>" + "<body><h1>" + "POST.
"</h1></body>" + "</html> ";
```

Show in Remote Systems view
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Team
Compare With
Configure
Source
Configure and Detect Nested Projects...
Convert to JPA Project...
Add Gradle Nature
Enable as Cloud Foundry App
Convert to Plug-in Projects...
Create module-info.java
Convert to Maven Project

Then in the generated pom.xml, add the following dependencies

```
<dependencies>

   <dependency>
   <groupId>org.glassfish.jersey.containers</groupId>
   <artifactId>jersey-container-servlet</artifactId>
   <version>2.40</version>
   </dependency>

   <dependency>
   <groupId>org.glassfish.jersey.inject</groupId>
   <artifactId>jersey-hk2</artifactId>
   <version>2.40</version>
    </dependency>

</dependencies>
```

Once the pom file is saved, the project should now download the dependencies jar files for you. You can also select *Maven → Update Project* to make sure the dependency jar files are updated. Now if you look at the Libraries → *Maven Dependencies*, you will see about 20 jars files downloaded.
Now, run the project on server. Use browser or curl to connect to the service. You should get the same result as in Exercise 1.

Exercise3: (**no submission, not graded**) import a plant sale REST project.
Download the provided *lab7PlantsRESTnew.zip*. Unzip it, then in eClipse, import it as Maven -> Existing Maven Project;
  Maven->Update project...
  Run as-> Run on Server
If the Java version of Project Facets does not match your system, change it to fit your system. Alternatively, you can always create a new dynamic project, convert it to Marven, and then copy the code and web.xml and pom.xml content to your project.

Look at the pom file. Compared with the pox.xml for exercise 2, this pom file has one more dependency

```
<dependency>
       <groupId>org.glassfish.jersey.media</groupId>
       <artifactId>jersey-media-json-jackson</artifactId>
       <version>2.40</version>
 </dependency>
```

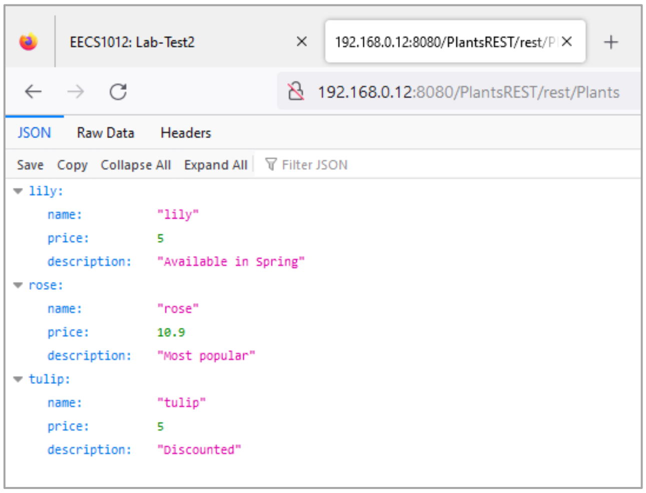
This is used to convert the returned types of various methods in PlantsforSale to json format.
Read and understand the code. Understand how different annotations (@path, @PathParam, @QueryParam, @DefaultValue) work, and how json format is used.
Use browser or curl to connect to the service.

Here are the example curl commands:

```
curl  http://localhost:8080/PlantsRESTnew/rest/Plants
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/rose
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/lily
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/tulip
curl -X POST  'http://localhost:8080/PlantsRESTnew/rest/Plants?id=green&plantName=green&price=2.3&desc=XXX'
curl -X POST  "http://localhost:8080/PlantsRESTnew/rest/Plants?id=blue&plantName=Blue&price=2.33 "
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/blue
curl  http://localhost:8080/PlantsRESTnew/rest/Plants/green
```

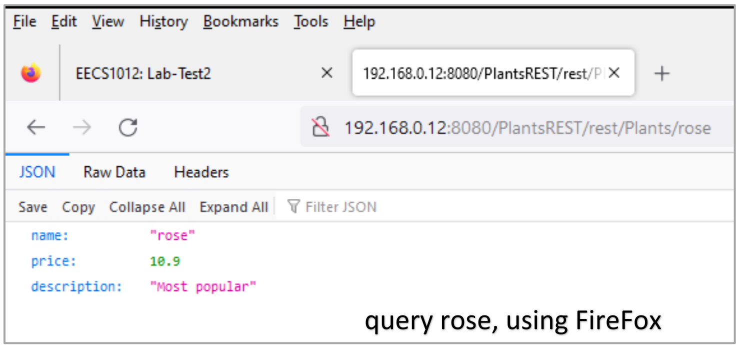
Note that for plant “blue”, it got default description “default desc” (why?)

You can connect using Postman or FireFox, which can display Json format nicely



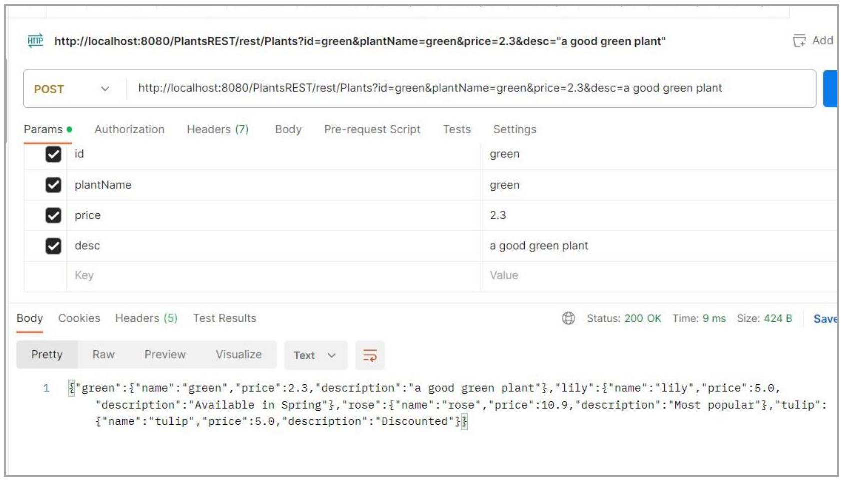
```
▼ lily:
    name:          "lily"
    price:         5
    description:   "Available in Spring"
▼ rose:
    name:          "rose"
    price:         10.9
    description:   "Most popular"
▼ tulip:
    name:          "tulip"
    price:         5
    description:   "Discounted"
```

query all Plants using FireFox



```
name:          "rose"
price:         10.9
description:   "Most popular"
```

query rose, using FireFox



http://localhost:8080/PlantsREST/rest/Plants?id=green&plantName=green&price=2.3&desc="a good green plant"

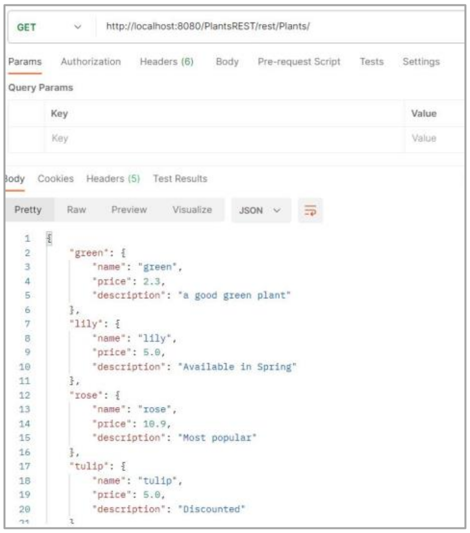
POST http://localhost:8080/PlantsREST/rest/Plants?id=green&plantName=green&price=2.3&desc=a good green plant

	Key	Value
☑	id	green
☑	plantName	green
☑	price	2.3
☑	desc	a good green plant

Status: 200 OK Time: 9 ms Size: 424 B

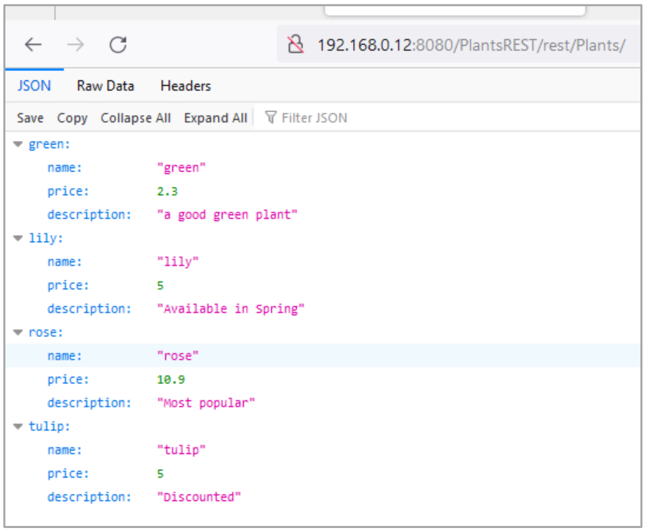
```
{"green":{"name":"green","price":2.3,"description":"a good green plant"},"lily":{"name":"lily","price":5.0,
"description":"Available in Spring"},"rose":{"name":"rose","price":10.9,"description":"Most popular"},"tulip":
{"name":"tulip","price":5.0,"description":"Discounted"}}
```

Add a resource, using postman



```
{
    "green": {
        "name": "green",
        "price": 2.3,
        "description": "a good green plant"
    },
    "lily": {
        "name": "lily",
        "price": 5.0,
        "description": "Available in Spring"
    },
    "rose": {
        "name": "rose",
        "price": 10.9,
        "description": "Most popular"
    },
    "tulip": {
        "name": "tulip",
        "price": 5.0,
        "description": "Discounted"
    }
```

query all plants, using postman



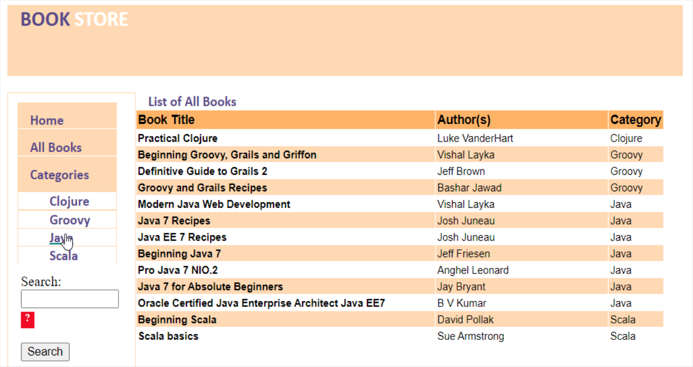
```
▼ green:
    name:          "green"
    price:         2.3
    description:   "a good green plant"
▼ lily:
    name:          "lily"
    price:         5
    description:   "Available in Spring"
▼ rose:
    name:          "rose"
    price:         10.9
    description:   "Most popular"
▼ tulip:
    name:          "tulip"
    price:         5
    description:   "Discounted"
```

query all plants, using FireFox

As a good practice, try to implement the update() and remove() functions, so that plants can be updated and removed.

Exercise 4 Evolve the book search application to a web service.

In Exercise 2 you developed a web app that allows users to search for books from database.



List of All Books

Book Title	Author(s)	Category
Practical Clojure	Luke VanderHart	Clojure
Beginning Groovy, Grails and Griffon	Vishal Layka	Groovy
Definitive Guide to Grails 2	Jeff Brown	Groovy
Groovy and Grails Recipes	Bashar Jawad	Groovy
Modern Java Web Development	Vishal Layka	Java
Java 7 Recipes	Josh Juneau	Java
Java EE 7 Recipes	Josh Juneau	Java
Beginning Java 7	Jeff Friesen	Java
Pro Java 7 NIO.2	Anghel Leonard	Java
Java 7 for Absolute Beginners	Jay Bryant	Java
Oracle Certified Java Enterprise Architect Java EE7	B V Kumar	Java
Beginning Scala	David Pollak	Scala
Scala basics	Sue Armstrong	Scala

Based on the code of exercise3 above, refactor the book database program so that it is a now RESTful web service.

You may want to create a dynamic web project **booksREST**, and convert it to Maven project and copy the pom.xml file and web.xml content from the Plant project to this project.

For this web service version, we don't need interface, the front controller class work with the DAO and Model classes. All the servlet and JSPs are not needed, and thus can be removed.

Following the REST style, the web service allows users to query categories, query all books, query books by category and by keyword.

Create a new package *restService*, in the package create a new class *BookService* (just a java class, not a servlet) as the front controller. Add several methods to support search all categories, search all books, search book by category, search book by keyword.

Use annotations to provide the various get methods.

Give service path */Books*.

For getting all books, use the service path */Books*. For getting all categories, use sub-path */category* For getting books by category, use path */searchByCat/{catid}* For getting books by keyword, use path */searchByKey/{keyword}.*

For this web service version, the front controller class work with the DAO and Model classes. All the servlet and JSPs are not needed, and thus can be removed.

As did in exercise 2, put the database file into the **WebContent** folder of your project, and use `getServletContext().getRealPath` to convert to absolute path.

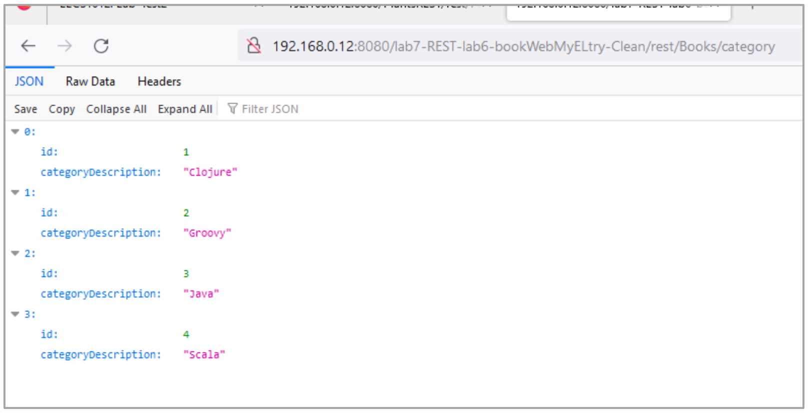
You may have the SQLite drive jar file in lib directory. As a practice, you can remove the jar file from *lib* folder, and add the following dependency in *pom.xml*

```
<dependency>
        <groupId>org.xerial</groupId>
        <artifactId>sqlite-jdbc</artifactId>
        <version>3.34.0</version>
</dependency>
```

When you save the file, or, *Maven -> Update Project*, a jdbc jar file will be downloaded for you, as you can see from *Maven Dependencies* folder. Clean the project and run it again.

Here are some curl commands

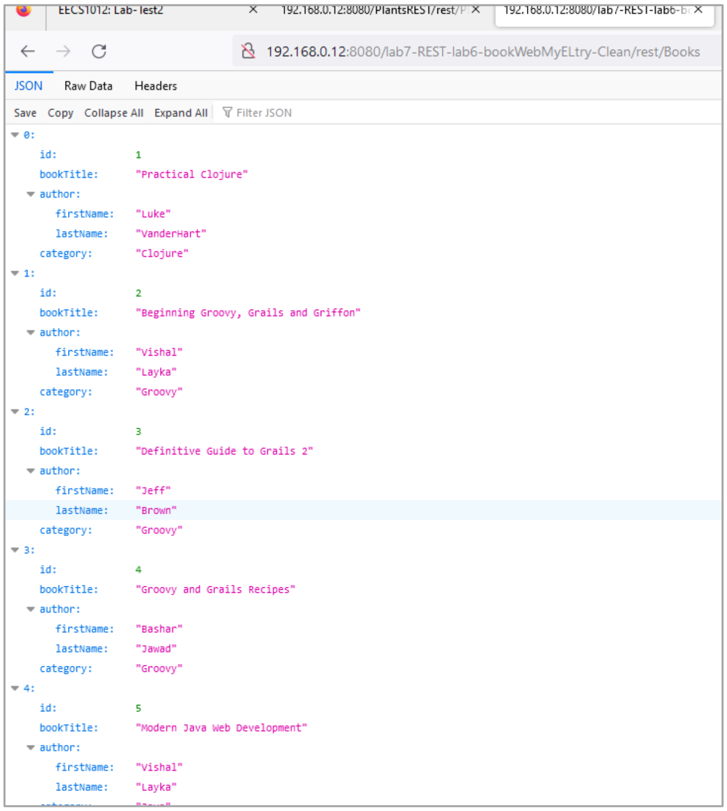
```
curl  http://localhost:8080/BooksREST/rest/Books
curl  http://localhost:8080/BooksREST/rest/Books/category
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Groovy
curl  http://localhost:8080/BooksREST/rest/Books/searchByCat/Scala
curl  http://localhost:8080/BooksREST/rest/Books/searchByKey/and
curl  http://localhost:8080/BooksREST/rest/Books/searchByKey/jeff
```



192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/category
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
▼ 0:
    id:                    1
    categoryDescription:   "Clojure"
▼ 1:
    id:                    2
    categoryDescription:   "Groovy"
▼ 2:
    id:                    3
    categoryDescription:   "Java"
▼ 3:
    id:                    4
    categoryDescription:   "Scala"
```

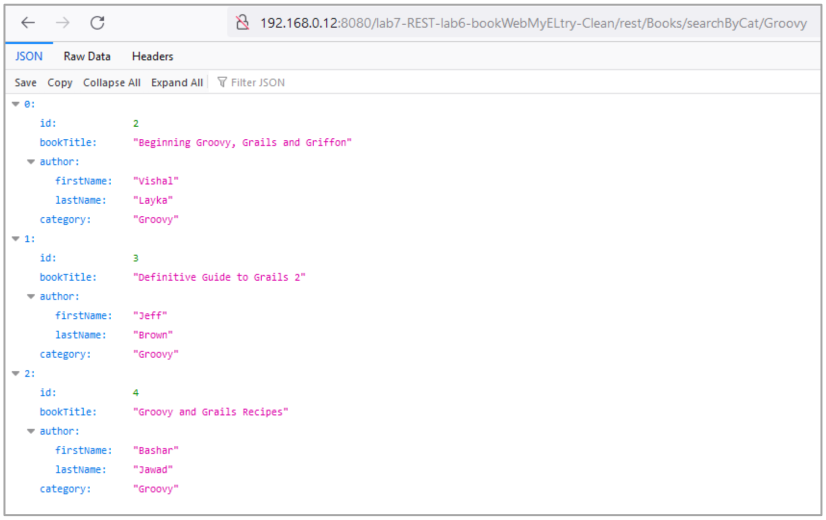
query all category, using FireFox



192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books

```
▼ 0:
    id:            1
    bookTitle:     "Practical Clojure"
  ▼ author:
      firstName:   "Luke"
      lastName:    "VanderHart"
    category:      "Clojure"
▼ 1:
    id:            2
    bookTitle:     "Beginning Groovy, Grails and Griffon"
  ▼ author:
      firstName:   "Vishal"
      lastName:    "Layka"
    category:      "Groovy"
▼ 2:
    id:            3
    bookTitle:     "Definitive Guide to Grails 2"
  ▼ author:
      firstName:   "Jeff"
      lastName:    "Brown"
    category:      "Groovy"
▼ 3:
    id:            4
    bookTitle:     "Groovy and Grails Recipes"
  ▼ author:
      firstName:   "Bashar"
      lastName:    "Jawad"
    category:      "Groovy"
▼ 4:
    id:            5
    bookTitle:     "Modern Java Web Development"
  ▼ author:
      firstName:   "Vishal"
      lastName:    "Layka"
    category:      "Java"
```

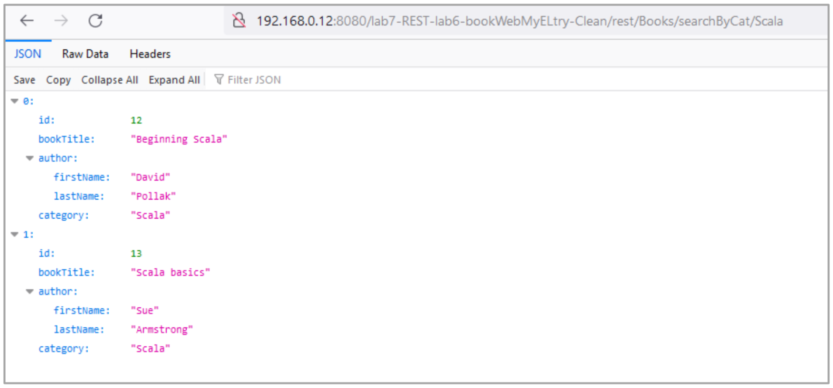
query all books, using FireFox



192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/searchByCat/Groovy

```
▼ 0:
    id:            2
    bookTitle:     "Beginning Groovy, Grails and Griffon"
  ▼ author:
      firstName:   "Vishal"
      lastName:    "Layka"
    category:      "Groovy"
▼ 1:
    id:            3
    bookTitle:     "Definitive Guide to Grails 2"
  ▼ author:
      firstName:   "Jeff"
      lastName:    "Brown"
    category:      "Groovy"
▼ 2:
    id:            4
    bookTitle:     "Groovy and Grails Recipes"
  ▼ author:
      firstName:   "Bashar"
      lastName:    "Jawad"
    category:      "Groovy"
```

query books by category, using FireFox

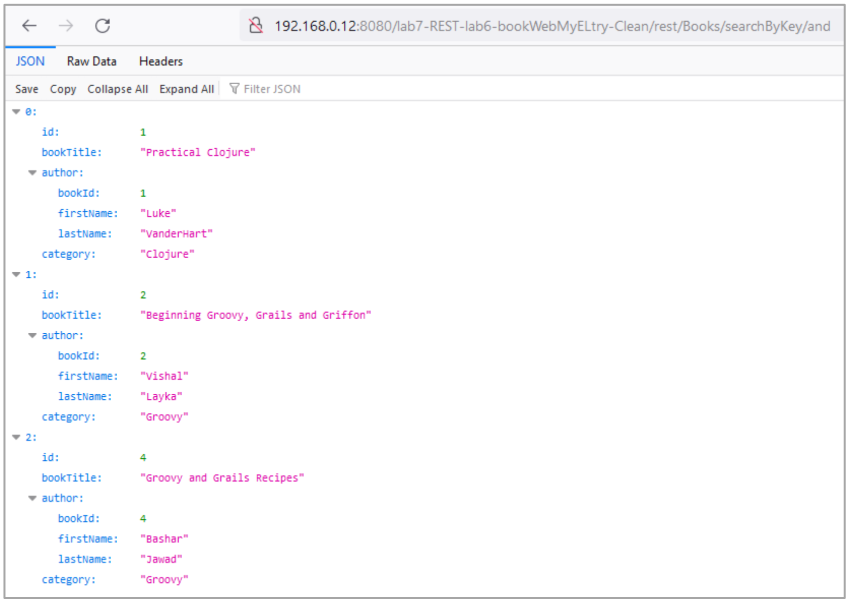


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/searchByCat/Scala

```
▼ 0:
    id:            12
    bookTitle:     "Beginning Scala"
  ▼ author:
      firstName:   "David"
      lastName:    "Pollak"
    category:      "Scala"
▼ 1:
    id:            13
    bookTitle:     "Scala basics"
  ▼ author:
      firstName:   "Sue"
      lastName:    "Armstrong"
    category:      "Scala"
```

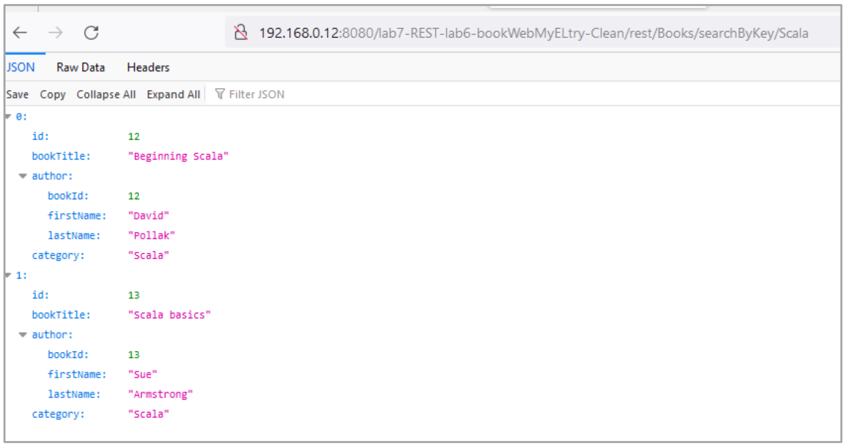
query books by category, using FireFox

..


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/searchByKey/and
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

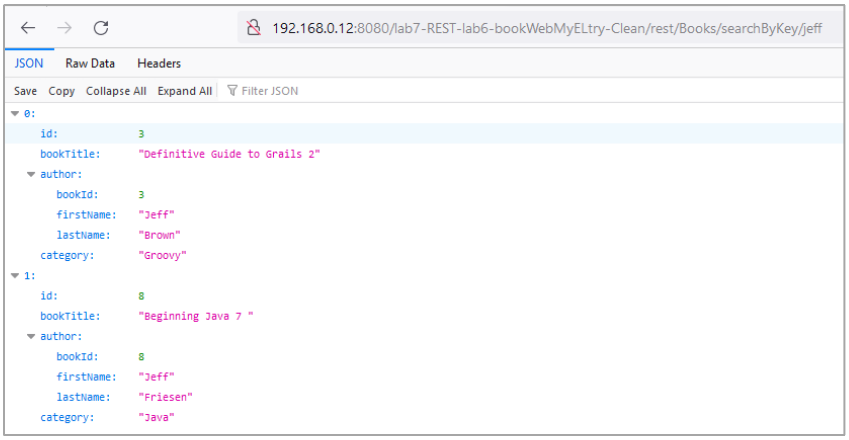
```
▼ 0:
    id:            1
    bookTitle:     "Practical Clojure"
  ▼ author:
      bookId:      1
      firstName:   "Luke"
      lastName:    "VanderHart"
    category:      "Clojure"
▼ 1:
    id:            2
    bookTitle:     "Beginning Groovy, Grails and Griffon"
  ▼ author:
      bookId:      2
      firstName:   "Vishal"
      lastName:    "Layka"
    category:      "Groovy"
▼ 2:
    id:            4
    bookTitle:     "Groovy and Grails Recipes"
  ▼ author:
      bookId:      4
      firstName:   "Bashar"
      lastName:    "Jawad"
    category:      "Groovy"
```

query books by keyword, using FireFox


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/searchByKey/Scala
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
▼ 0:
    id:            12
    bookTitle:     "Beginning Scala"
  ▼ author:
      bookId:      12
      firstName:   "David"
      lastName:    "Pollak"
    category:      "Scala"
▼ 1:
    id:            13
    bookTitle:     "Scala basics"
  ▼ author:
      bookId:      13
      firstName:   "Sue"
      lastName:    "Armstrong"
    category:      "Scala"
```

query books by keyword, using FireFox


192.168.0.12:8080/lab7-REST-lab6-bookWebMyELtry-Clean/rest/Books/searchByKey/jeff
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
▼ 0:
    id:            3
    bookTitle:     "Definitive Guide to Grails 2"
  ▼ author:
      bookId:      3
      firstName:   "Jeff"
      lastName:    "Brown"
    category:      "Groovy"
▼ 1:
    id:            8
    bookTitle:     "Beginning Java 7 "
  ▼ author:
      bookId:      8
      firstName:   "Jeff"
      lastName:    "Friesen"
    category:      "Java"
```

query books by keyword, using FireFox

- Submit: for this maven project, to make sure the TA can run your code, export your project in two ways and submit both:
  1. export as war file with default name **booksREST.war**. Make sure to check 'export source files'
  2. export as *Archive File* (General → Archive File), with name **booksREST.zip**.

  Please submit both the war file and the zip file for this question.

In summary, in this lab you submit the following files:
lab6servletQ1.war
lab6servletQ2.war
booksREST.war
booksREST.zip

For the war files, Make sure you have checked “Export source files” when exporting.

Submit the files separately on eClass.

Late submissions or submissions by email will NOT be accepted. Plan ahead and submit early.

Submissions.

You should have exported **lab6servletQ1.war lab6servletQ2.war**. Make sure you have checked “Export source files” when exporting.

Submit the 2 war files separately on eClass.

Late submissions or submissions by email will NOT be accepted. Plan ahead and submit early.