<https://www.javaguides.net/2019/03/jsp-servlet-jdbc-mysql-crud-example-tutorial.html>

### JSP Servlet JDBC MySQL CRUD Example Tutorial

**author:**[**Ramesh Fadatare**](https://www.blogger.com/profile/14691512106162803120)

In this tutorial, we are building a simple **User Management** web application that manages a collection of **users** with the basic feature: list, insert, update, delete (or CURD operations - Create, Update, Read and Delete).

You can download the source code of this tutorial from my GitHub repository and the link is given at the end of this tutorial.

Top JSP, Servlet and JDBC Tutorials:

* [**Servlet Tutorial**](https://www.javaguides.net/p/servlet-tutorial.html)
* [**JSP Tutorial**](https://www.javaguides.net/p/jsp-tutorial.html)
* [**JDBC 4.2 Tutorial**](https://www.javaguides.net/p/jdbc-tutorial.html)

Check out [**Build Todo App using JSP, Servlet, JDBC, and MySQL**](https://www.javaguides.net/2019/10/build-todo-app-using-jsp-servlet-jdbc-and-mysql.html).

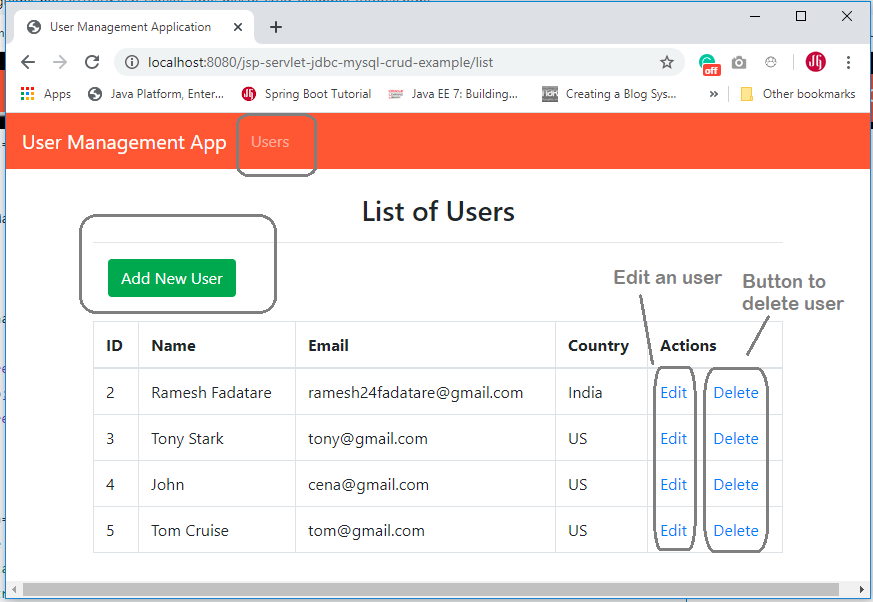
## Video Tutorial

This tutorial explained very well in below youtube video. Subscribe to our youtube channel for more future video updates.

We will develop below simple basic features in our **User Management** web application:

1. Create a User
2. Update a User
3. Delete a User
4. Retrieve a User
5. List of all Users

The application looks something like this:

**[](https://1.bp.blogspot.com/-V7u_EKarkws/XiRUgEgUD9I/AAAAAAAAHYA/Sh8GLesf0cAaVoxNVx7C-qRuQtN1hd5NACLcBGAsYHQ/s1600/list-users.PNG)**

## Tools and technologies used

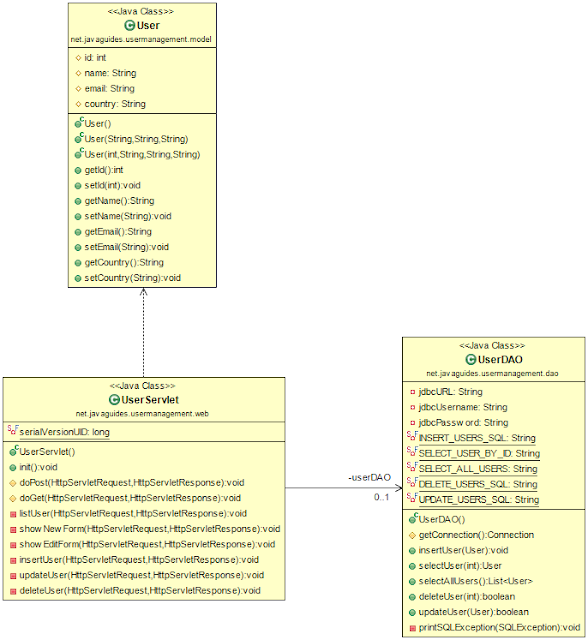
* JSP - 2.2 +
* IDE - STS/Eclipse Neon.3
* JDK - 1.8 or later
* Apache Tomcat - 8.5
* JSTL - 1.2.1
* Servlet API - 2.5
* MySQL - mysql-connector-java-8.0.13.jar

## Development Steps

1. Create an Eclipse Dynamic Web Project
2. Add Dependencies
3. Project Structure
4. MySQL Database Setup
5. Create a JavaBean - User.java
6. Create a UserDAO.java
7. Create a UserServlet.java
8. Creating User Listing JSP Page - user-list.jsp
9. Create a User Form JSP Page - user-form.jsp
10. Creating Error JSP page
11. Deploying and Testing the Application Demo

## Class Diagram

Here is the class diagram of the **User Management** web application that we are going to develop in this tutorial:

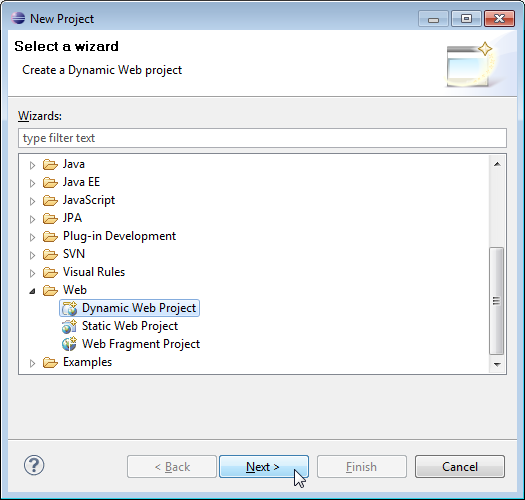
**[](https://1.bp.blogspot.com/-TTRbvhKD8zM/XiRO_h7SQMI/AAAAAAAAHX0/c9h1229nNhEgAcgWz_LRCe1UsWb5PwmxgCLcBGAsYHQ/s1600/jsp-servlet-jdbc-crud-class-diagram.png)**

## 1. Create an Eclipse Dynamic Web Project

To create a new dynamic Web project in Eclipse:

1. On the main menu select **File > New > Project....**

2. In the upcoming wizard choose **Web > Dynamic Web Project.**

**[](https://3.bp.blogspot.com/-CeA278XIG4g/XDiv1ePWVlI/AAAAAAAAFZI/2mvCujNS2Co_gp1UqevtbZk0qhDsaa9DwCLcBGAs/s1600/create-web-proj-1.png)**

3. Click **Next**.

4. Enter project name as "jsp-servlet-jdbc-mysql-example";  
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

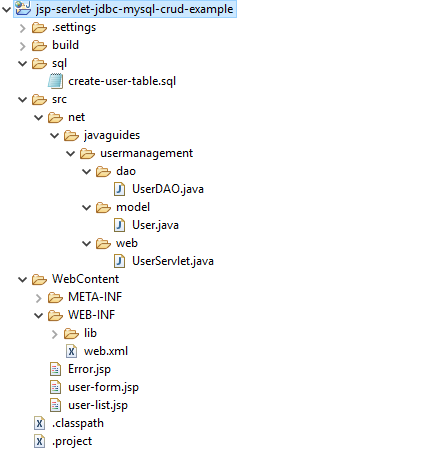
## 2. Add Dependencies

Add the latest release of below jar files to the lib folder.

* jsp-api.2.3.1.jar
* servlet-api.2.3.jar
* mysql-connector-java-8.0.13.jar
* jstl-1.2.jar

## 3. Project Structure

Standard project structure for your reference -

**[](https://1.bp.blogspot.com/-jviGGrVkagY/XISpZULFm8I/AAAAAAAAFvg/cRyR_P5mezgq71W3n-B4hO1XE-tQ0V57ACLcBGAs/s1600/project-structure.PNG)**

## 4. MySQL Database Setup

Let's create a database named "demo" in MySQL. Now, create a users table using below DDL script:

CREATE DATABASE 'demo';

USE demo;

create table users (

id int(3) NOT NULL AUTO\_INCREMENT,

name varchar(120) NOT NULL,

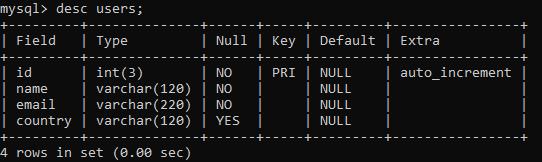
email varchar(220) NOT NULL,

country varchar(120),

PRIMARY KEY (id)

);

You can use either MySQL Command Line Client or MySQL Workbench tool to create the database. The above a **users**table looks like:

**[](https://1.bp.blogspot.com/-o4y42hluC-s/XISptw_7GII/AAAAAAAAFvo/Qk7zd0yD2F8pywWwOjHX_JJHz0HoA0tcwCLcBGAs/s1600/users-mysql-table.PNG)**

## 5. Create a JavaBean - User.java

Let's create a *User* java class to model a **user**entity in the database with the following code:

package net.javaguides.usermanagement.model;

/\*\*

\* User.java

\* This is a model class represents a User entity

\* @author Ramesh Fadatare

\*

\*/

public class User {

protected int id;

protected String name;

protected String email;

protected String country;

public User() {}

public User(String name, String email, String country) {

super();

this.name = name;

this.email = email;

this.country = country;

}

public User(int id, String name, String email, String country) {

super();

this.id = id;

this.name = name;

this.email = email;

this.country = country;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getCountry() {

return country;

}

public void setCountry(String country) {

this.country = country;

}

}

## 6. Create a UserDAO.java

Let's create a *UserDAO* class which is a Data Access Layer (DAO) class that provides CRUD (Create, Read, Update, Delete) operations for the table **users**in a database. Here’s the full source code of the *UserDAO*:

package net.javaguides.usermanagement.dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import net.javaguides.usermanagement.model.User;

/\*\*

\* AbstractDAO.java This DAO class provides CRUD database operations for the

\* table users in the database.

\*

\* @author Ramesh Fadatare

\*

\*/

public class UserDAO {

private String jdbcURL = "jdbc:mysql://localhost:3306/demo?useSSL=false";

private String jdbcUsername = "root";

private String jdbcPassword = "root";

private static final String INSERT\_USERS\_SQL = "INSERT INTO users" + " (name, email, country) VALUES " +

" (?, ?, ?);";

private static final String SELECT\_USER\_BY\_ID = "select id,name,email,country from users where id =?";

private static final String SELECT\_ALL\_USERS = "select \* from users";

private static final String DELETE\_USERS\_SQL = "delete from users where id = ?;";

private static final String UPDATE\_USERS\_SQL = "update users set name = ?,email= ?, country =? where id = ?;";

public UserDAO() {}

protected Connection getConnection() {

Connection connection = null;

try {

Class.forName("com.mysql.jdbc.Driver");

connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return connection;

}

public void insertUser(User user) throws SQLException {

System.out.println(INSERT\_USERS\_SQL);

// try-with-resource statement will auto close the connection.

try (Connection connection = getConnection(); PreparedStatement preparedStatement = connection.prepareStatement(INSERT\_USERS\_SQL)) {

preparedStatement.setString(1, user.getName());

preparedStatement.setString(2, user.getEmail());

preparedStatement.setString(3, user.getCountry());

System.out.println(preparedStatement);

preparedStatement.executeUpdate();

} catch (SQLException e) {

printSQLException(e);

}

}

public User selectUser(int id) {

User user = null;

// Step 1: Establishing a Connection

try (Connection connection = getConnection();

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(SELECT\_USER\_BY\_ID);) {

preparedStatement.setInt(1, id);

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

// Step 4: Process the ResultSet object.

while (rs.next()) {

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

user = new User(id, name, email, country);

}

} catch (SQLException e) {

printSQLException(e);

}

return user;

}

public List < User > selectAllUsers() {

// using try-with-resources to avoid closing resources (boiler plate code)

List < User > users = new ArrayList < > ();

// Step 1: Establishing a Connection

try (Connection connection = getConnection();

// Step 2:Create a statement using connection object

PreparedStatement preparedStatement = connection.prepareStatement(SELECT\_ALL\_USERS);) {

System.out.println(preparedStatement);

// Step 3: Execute the query or update query

ResultSet rs = preparedStatement.executeQuery();

// Step 4: Process the ResultSet object.

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name");

String email = rs.getString("email");

String country = rs.getString("country");

users.add(new User(id, name, email, country));

}

} catch (SQLException e) {

printSQLException(e);

}

return users;

}

public boolean deleteUser(int id) throws SQLException {

boolean rowDeleted;

try (Connection connection = getConnection(); PreparedStatement statement = connection.prepareStatement(DELETE\_USERS\_SQL);) {

statement.setInt(1, id);

rowDeleted = statement.executeUpdate() > 0;

}

return rowDeleted;

}

public boolean updateUser(User user) throws SQLException {

boolean rowUpdated;

try (Connection connection = getConnection(); PreparedStatement statement = connection.prepareStatement(UPDATE\_USERS\_SQL);) {

statement.setString(1, user.getName());

statement.setString(2, user.getEmail());

statement.setString(3, user.getCountry());

statement.setInt(4, user.getId());

rowUpdated = statement.executeUpdate() > 0;

}

return rowUpdated;

}

private void printSQLException(SQLException ex) {

for (Throwable e: ex) {

if (e instanceof SQLException) {

e.printStackTrace(System.err);

System.err.println("SQLState: " + ((SQLException) e).getSQLState());

System.err.println("Error Code: " + ((SQLException) e).getErrorCode());

System.err.println("Message: " + e.getMessage());

Throwable t = ex.getCause();

while (t != null) {

System.out.println("Cause: " + t);

t = t.getCause();

}

}

}

}

}

## 7. Create a UserServlet.java

Now, let's create *UserServlet* that acts as a page controller to handle all requests from the client. Let’s look at the code first:

package net.javaguides.usermanagement.web;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import net.javaguides.usermanagement.dao.UserDAO;

import net.javaguides.usermanagement.model.User;

/\*\*

\* ControllerServlet.java

\* This servlet acts as a page controller for the application, handling all

\* requests from the user.

\* @email Ramesh Fadatare

\*/

@WebServlet("/")

public class UserServlet extends HttpServlet {

private static final long serialVersionUID = 1 L;

private UserDAO userDAO;

public void init() {

userDAO = new UserDAO();

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

doGet(request, response);

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String action = request.getServletPath();

try {

switch (action) {

case "/new":

showNewForm(request, response);

break;

case "/insert":

insertUser(request, response);

break;

case "/delete":

deleteUser(request, response);

break;

case "/edit":

showEditForm(request, response);

break;

case "/update":

updateUser(request, response);

break;

default:

listUser(request, response);

break;

}

} catch (SQLException ex) {

throw new ServletException(ex);

}

}

private void listUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException, ServletException {

List < User > listUser = userDAO.selectAllUsers();

request.setAttribute("listUser", listUser);

RequestDispatcher dispatcher = request.getRequestDispatcher("user-list.jsp");

dispatcher.forward(request, response);

}

private void showNewForm(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");

dispatcher.forward(request, response);

}

private void showEditForm(HttpServletRequest request, HttpServletResponse response)

throws SQLException, ServletException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

User existingUser = userDAO.selectUser(id);

RequestDispatcher dispatcher = request.getRequestDispatcher("user-form.jsp");

request.setAttribute("user", existingUser);

dispatcher.forward(request, response);

}

private void insertUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

String name = request.getParameter("name");

String email = request.getParameter("email");

String country = request.getParameter("country");

User newUser = new User(name, email, country);

userDAO.insertUser(newUser);

response.sendRedirect("list");

}

private void updateUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

String name = request.getParameter("name");

String email = request.getParameter("email");

String country = request.getParameter("country");

User book = new User(id, name, email, country);

userDAO.updateUser(book);

response.sendRedirect("list");

}

private void deleteUser(HttpServletRequest request, HttpServletResponse response)

throws SQLException, IOException {

int id = Integer.parseInt(request.getParameter("id"));

userDAO.deleteUser(id);

response.sendRedirect("list");

}

}

## 8. Creating User Listing JSP Page - user-list.jsp

Next, create a JSP page for displaying all **users**from the database. Let's create a *list-user.jsp* page under the **WebContent** directory in the project with the following code:

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<html>

<head>

<title>User Management Application</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<header>

<nav class="navbar navbar-expand-md navbar-dark" style="background-color: tomato">

<div>

<a href="https://www.javaguides.net" class="navbar-brand"> User

Management App </a>

</div>

<ul class="navbar-nav">

<li><a href="<%=request.getContextPath()%>/list" class="nav-link">Users</a></li>

</ul>

</nav>

</header>

<br>

<div class="row">

<!-- <div class="alert alert-success" \*ngIf='message'>{{message}}</div> -->

<div class="container">

<h3 class="text-center">List of Users</h3>

<hr>

<div class="container text-left">

<a href="<%=request.getContextPath()%>/new" class="btn btn-success">Add

New User</a>

</div>

<br>

<table class="table table-bordered">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

<th>Country</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

<!-- for (Todo todo: todos) { -->

<c:forEach var="user" items="${listUser}">

<tr>

<td>

<c:out value="${user.id}" />

</td>

<td>

<c:out value="${user.name}" />

</td>

<td>

<c:out value="${user.email}" />

</td>

<td>

<c:out value="${user.country}" />

</td>

<td><a href="edit?id=<c:out value='${user.id}' />">Edit</a> &nbsp;&nbsp;&nbsp;&nbsp; <a href="delete?id=<c:out value='${user.id}' />">Delete</a></td>

</tr>

</c:forEach>

<!-- } -->

</tbody>

</table>

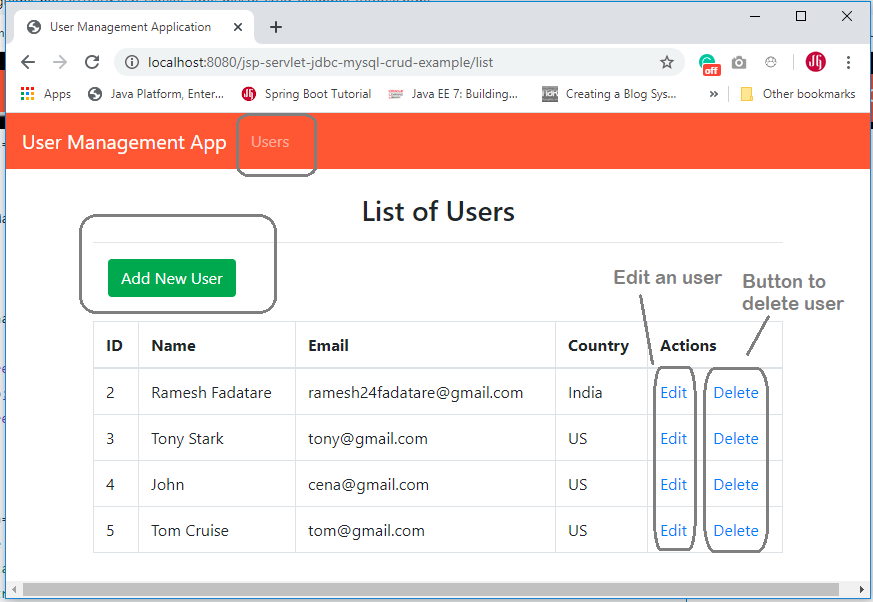
</div>

</div>

</body>

</html>

Once you will deploy above JSP page in tomcat and open in the browser looks something like this:

**[](https://1.bp.blogspot.com/-V7u_EKarkws/XiRUgEgUD9I/AAAAAAAAHYA/Sh8GLesf0cAaVoxNVx7C-qRuQtN1hd5NACLcBGAsYHQ/s1600/list-users.PNG)**

## 9. Create a User Form JSP Page - user-form.jsp

Next, we create a JSP page for creating a new User called *user-form.jsp*. Here’s its full source code:

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<html>

<head>

<title>User Management Application</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

</head>

<body>

<header>

<nav class="navbar navbar-expand-md navbar-dark" style="background-color: tomato">

<div>

<a href="https://www.javaguides.net" class="navbar-brand"> User Management App </a>

</div>

<ul class="navbar-nav">

<li><a href="<%=request.getContextPath()%>/list" class="nav-link">Users</a></li>

</ul>

</nav>

</header>

<br>

<div class="container col-md-5">

<div class="card">

<div class="card-body">

<c:if test="${user != null}">

<form action="update" method="post">

</c:if>

<c:if test="${user == null}">

<form action="insert" method="post">

</c:if>

<caption>

<h2>

<c:if test="${user != null}">

Edit User

</c:if>

<c:if test="${user == null}">

Add New User

</c:if>

</h2>

</caption>

<c:if test="${user != null}">

<input type="hidden" name="id" value="<c:out value='${user.id}' />" />

</c:if>

<fieldset class="form-group">

<label>User Name</label> <input type="text" value="<c:out value='${user.name}' />" class="form-control" name="name" required="required">

</fieldset>

<fieldset class="form-group">

<label>User Email</label> <input type="text" value="<c:out value='${user.email}' />" class="form-control" name="email">

</fieldset>

<fieldset class="form-group">

<label>User Country</label> <input type="text" value="<c:out value='${user.country}' />" class="form-control" name="country">

</fieldset>

<button type="submit" class="btn btn-success">Save</button>

</form>

</div>

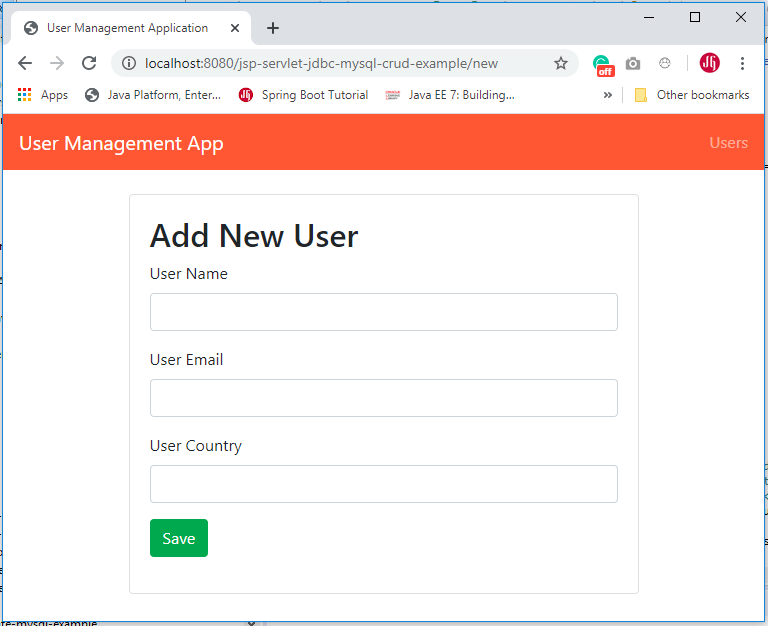
</div>

</div>

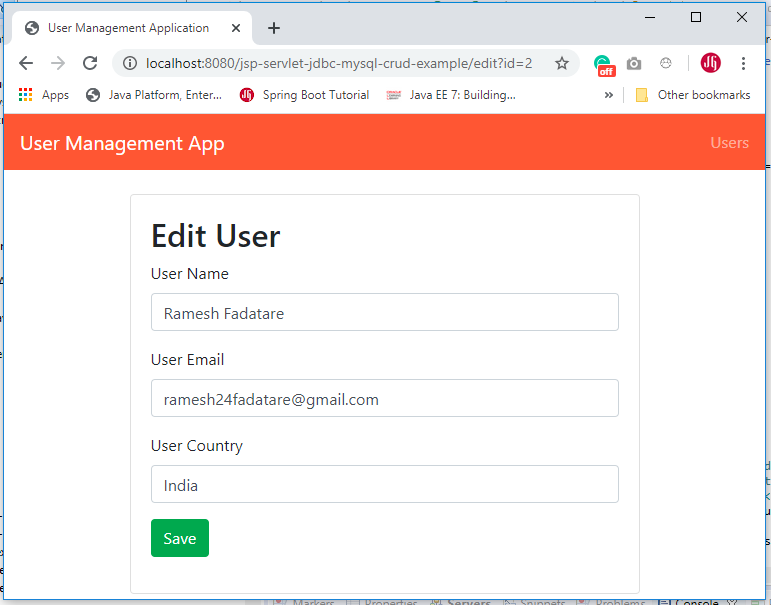
</body>

</html>

Once you will deploy above JSP page in tomcat and open in the browser looks something like this:

**[](https://1.bp.blogspot.com/-krK2NUKZNvY/XiRU7pihv7I/AAAAAAAAHYI/MLpBBT-eT5AItc0ol4D3dL7gzwCLv5p8wCLcBGAsYHQ/s1600/add-user.PNG)**

The above page acts for both functionalities to create a new User and Edit the same user. The edit page looks like:

**[](https://1.bp.blogspot.com/-Ef8GHKpoxdU/XiRVBRAGfyI/AAAAAAAAHYM/rSy2v0onDQQpmpe3cPk3JPfK8edS0TF_gCLcBGAsYHQ/s1600/edit-user.PNG)**

## 10. Creating Error JSP page

Here’s the code of the *Error.jsp* page which simply shows the exception message:

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8" isErrorPage="true" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<title>Error</title>

</head>

<body>

<center>

<h1>Error</h1>

<h2><%=exception.getMessage() %><br/> </h2>

</center>

</body>

</html>

## 11. Deploying and Testing the Application

It's time to see a demo of the above **User Management** web application. Deploy this web application in tomcat server.

Type the following URL in your web browser to access the **User Management**application: [**http://localhost:8080/jsp-servlet-jdbc-mysql-crud-example/**](http://localhost:8080/jsp-servlet-jdbc-mysql-crud-example/)

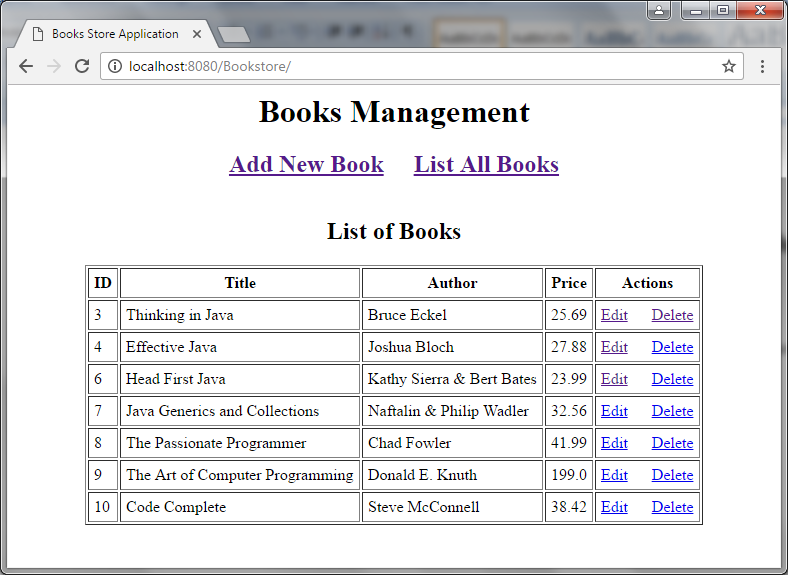
<https://www.codejava.net/coding/jsp-servlet-jdbc-mysql-create-read-update-delete-crud-example>

# JSP Servlet JDBC MySQL Create Read Update Delete (CRUD) Example

Written by  [Nam Ha Minh](https://www.codejava.net/nam-ha-minh)

Last Updated on 05 November 2023   |  [Print](https://www.codejava.net/coding/jsp-servlet-jdbc-mysql-create-read-update-delete-crud-example?tmpl=component&print=1&page=)[Email](https://www.codejava.net/component/mailto/?tmpl=component&template=protostar&link=f1c5ffbb986fd748b53db2a60b2b50da0501e1e5)

In this Java tutorial, we’re going to help you understand the process of coding a basic Java web application that manages a collection of books with the basic feature: list, insert, update, delete (or CURD operations - Create, Update, Read and Delete). The application looks something like this:



You will learn to how to build this application using the following technologies:

* Java Servlets and Java Server Pages (JSP)
* JSP Standard Tag Library (JSTL)
* Java Database Connectivity (JDBC)
* MySQL database
* Apache Tomcat Server

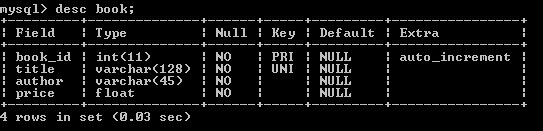
We use Eclipse IDE with Maven to develop the project.

## 1. Creating MySQL Database

For simplicity, we have only one table. Execute the following MySQL script to create a database named **Bookstore**and a table named **Book**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | CREATE DATABASE 'Bookstore';  USE Bookstore;    CREATE TABLE `book` (    `book\_id` int(11) NOT NULL AUTO\_INCREMENT,    `title` varchar(128) NOT NULL,    `author` varchar(45) NOT NULL,    `price` float NOT NULL,    PRIMARY KEY (`book\_id`),    UNIQUE KEY `book\_id\_UNIQUE` (`book\_id`),    UNIQUE KEY `title\_UNIQUE` (`title`)  ) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=latin1 |

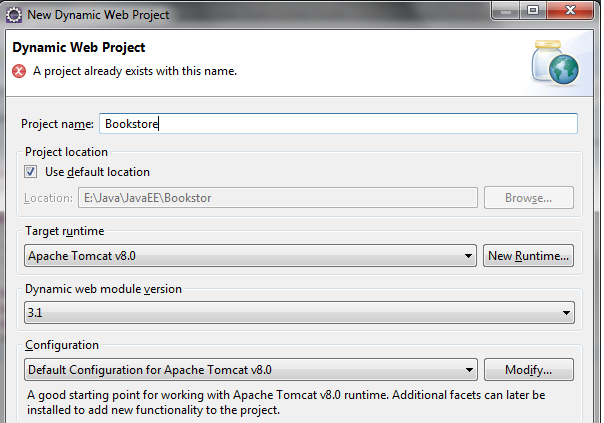
The table **book** has structure like this:



You can use either MySQL Command Line Client or MySQL Workbench tool to create the database.

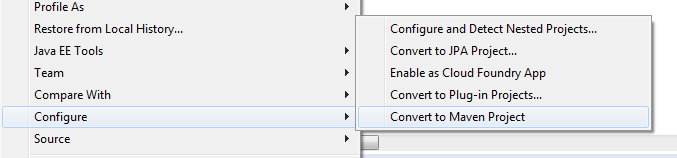
## 2. Creating Eclipse Project with Maven

In Eclipse IDE, click **File > New > Dynamic Web Project** to create a new Java dynamic web project. Name the project as Bookstore:



Remember to choose **Target runtime** as *Apache Tomcat v8.0* and **Dynamic web module version** as *3.1* (this is the Java servlet version).

Click **Finish**. Then convert this project to a Maven project by right click on the project, select **Configure > Convert to Maven Project**, as shown below:



You need to enter information to create Maven POM file, such as group ID, artifact ID, etc. Then add the following dependencies to the **pom.xml**file:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <dependencies>      <dependency>          <groupId>javax.servlet</groupId>          <artifactId>javax.servlet-api</artifactId>          <version>3.1.0</version>          <scope>provided</scope>      </dependency>      <dependency>          <groupId>javax.servlet.jsp</groupId>          <artifactId>javax.servlet.jsp-api</artifactId>          <version>2.3.1</version>          <scope>provided</scope>      </dependency>      <dependency>          <groupId>jstl</groupId>          <artifactId>jstl</artifactId>          <version>1.2</version>      </dependency>      <dependency>          <groupId>mysql</groupId>          <artifactId>mysql-connector-java</artifactId>          <version>5.1.30</version>      </dependency>  </dependencies> |

As you can see, the dependencies here are for Servlet, JSP, JSTL and MySQL connector Java (a JDBC driver for MySQL).

And remember to create a Java package for the project, here we use the package name **net.codejava.javaee.bookstore**.

## 3. Writing Model Class

Next, create a Java class named **Book.java** to model a book entity in the database with the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | package net.codejava.javaee.bookstore;    /\*\*   \* Book.java   \* This is a model class represents a book entity   \* @author www.codejava.net   \*   \*/  public class Book {      protected int id;      protected String title;      protected String author;      protected float price;        public Book() {      }        public Book(int id) {          this.id = id;      }        public Book(int id, String title, String author, float price) {          this(title, author, price);          this.id = id;      }        public Book(String title, String author, float price) {          this.title = title;          this.author = author;          this.price = price;      }        public int getId() {          return id;      }        public void setId(int id) {          this.id = id;      }        public String getTitle() {          return title;      }        public void setTitle(String title) {          this.title = title;      }        public String getAuthor() {          return author;      }        public void setAuthor(String author) {          this.author = author;      }        public float getPrice() {          return price;      }        public void setPrice(float price) {          this.price = price;      }  } |

As you can see, this class has 4 fields according to 4 columns in the table **book** in database: id, title, author and price.

[](https://www.udemy.com/course/java-servlet-jsp-and-hibernate-build-a-complete-website/?couponCode=CJA102)

[**Java Servlet, JSP and Hibernate: Build eCommerce Website**](https://www.udemy.com/course/java-servlet-jsp-and-hibernate-build-a-complete-website/?couponCode=CJA102)

***Learn Java Servlet, JSP, Hibernate framework to build an eCommerce Website (with PayPal and credit card payment)***

## 4. Coding DAO class

Next, we need to implement a Data Access Layer (DAO) class that provides CRUD (Create, Read, Update, Delete) operations for the table **book** in database. Here’s the full source code of the **BookDAO**class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147 | package net.codejava.javaee.bookstore;    import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;  import java.util.ArrayList;  import java.util.List;    /\*\*   \* AbstractDAO.java   \* This DAO class provides CRUD database operations for the table book   \* in the database.   \* @author www.codejava.net   \*   \*/  public class BookDAO {      private String jdbcURL;      private String jdbcUsername;      private String jdbcPassword;      private Connection jdbcConnection;        public BookDAO(String jdbcURL, String jdbcUsername, String jdbcPassword) {          this.jdbcURL = jdbcURL;          this.jdbcUsername = jdbcUsername;          this.jdbcPassword = jdbcPassword;      }        protected void connect() throws SQLException {          if (jdbcConnection == null || jdbcConnection.isClosed()) {              try {                  Class.forName("com.mysql.jdbc.Driver");              } catch (ClassNotFoundException e) {                  throw new SQLException(e);              }              jdbcConnection = DriverManager.getConnection(                                          jdbcURL, jdbcUsername, jdbcPassword);          }      }        protected void disconnect() throws SQLException {          if (jdbcConnection != null && !jdbcConnection.isClosed()) {              jdbcConnection.close();          }      }        public boolean insertBook(Book book) throws SQLException {          String sql = "INSERT INTO book (title, author, price) VALUES (?, ?, ?)";          connect();            PreparedStatement statement = jdbcConnection.prepareStatement(sql);          statement.setString(1, book.getTitle());          statement.setString(2, book.getAuthor());          statement.setFloat(3, book.getPrice());            boolean rowInserted = statement.executeUpdate() > 0;          statement.close();          disconnect();          return rowInserted;      }        public List<Book> listAllBooks() throws SQLException {          List<Book> listBook = new ArrayList<>();            String sql = "SELECT \* FROM book";            connect();            Statement statement = jdbcConnection.createStatement();          ResultSet resultSet = statement.executeQuery(sql);            while (resultSet.next()) {              int id = resultSet.getInt("book\_id");              String title = resultSet.getString("title");              String author = resultSet.getString("author");              float price = resultSet.getFloat("price");                Book book = new Book(id, title, author, price);              listBook.add(book);          }            resultSet.close();          statement.close();            disconnect();            return listBook;      }        public boolean deleteBook(Book book) throws SQLException {          String sql = "DELETE FROM book where book\_id = ?";            connect();            PreparedStatement statement = jdbcConnection.prepareStatement(sql);          statement.setInt(1, book.getId());            boolean rowDeleted = statement.executeUpdate() > 0;          statement.close();          disconnect();          return rowDeleted;      }        public boolean updateBook(Book book) throws SQLException {          String sql = "UPDATE book SET title = ?, author = ?, price = ?";          sql += " WHERE book\_id = ?";          connect();            PreparedStatement statement = jdbcConnection.prepareStatement(sql);          statement.setString(1, book.getTitle());          statement.setString(2, book.getAuthor());          statement.setFloat(3, book.getPrice());          statement.setInt(4, book.getId());            boolean rowUpdated = statement.executeUpdate() > 0;          statement.close();          disconnect();          return rowUpdated;      }        public Book getBook(int id) throws SQLException {          Book book = null;          String sql = "SELECT \* FROM book WHERE book\_id = ?";            connect();            PreparedStatement statement = jdbcConnection.prepareStatement(sql);          statement.setInt(1, id);            ResultSet resultSet = statement.executeQuery();            if (resultSet.next()) {              String title = resultSet.getString("title");              String author = resultSet.getString("author");              float price = resultSet.getFloat("price");                book = new Book(id, title, author, price);          }            resultSet.close();          statement.close();            return book;      }  } |

As you can see, the JDBC connection information is injected to this class via its constructor. And the following methods are for CRUD operations:

* Create: **insertBook(Book)**- this inserts a new row into the table **book**.
* Read: **listAllBooks()** - this retrieves all rows; and **getBook(id)**- returns a specific row based on the primary key value (ID).
* Update: **updateBook(Book)**- this updates an existing row in the database.
* Delete: **deleteBook(Book)** - this removes an existing row in the database based on the primary key value (ID).

For detailed instructions on CRUD operations with JDBC, see JDBC Tutorial: [SQL Insert, Select, Update, and Delete Examples](https://www.codejava.net/java-se/jdbc/jdbc-tutorial-sql-insert-select-update-and-delete-examples).

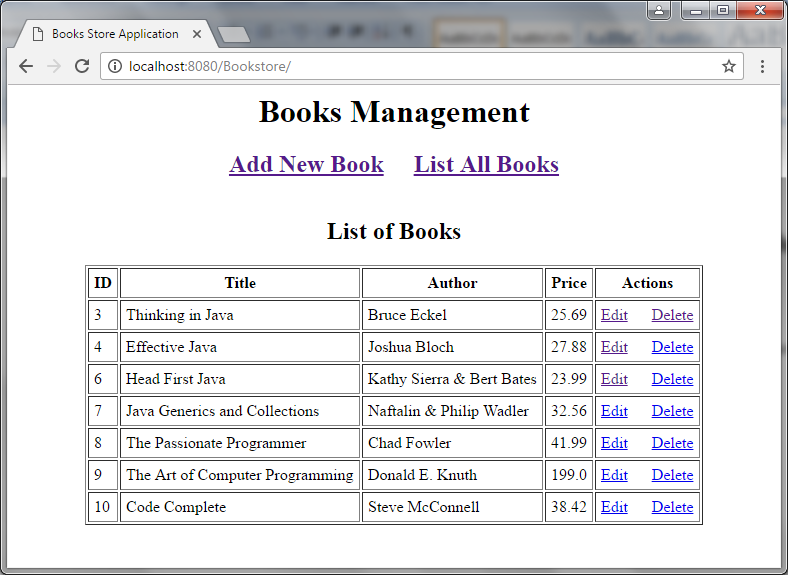
## 5. Writing Book Listing JSP Page

Next, create a JSP page for displaying all books from the database. The following is code of the **BookList.jsp**page under the **WebContent**directory in the project:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <html>  <head>      <title>Books Store Application</title>  </head>  <body>      <center>          <h1>Books Management</h1>          <h2>              <a href="/new">Add New Book</a>              &nbsp;&nbsp;&nbsp;              <a href="/list">List All Books</a>            </h2>      </center>      <div align="center">          <table border="1" cellpadding="5">              <caption><h2>List of Books</h2></caption>              <tr>                  <th>ID</th>                  <th>Title</th>                  <th>Author</th>                  <th>Price</th>                  <th>Actions</th>              </tr>              <c:forEach var="book" items="${listBook}">                  <tr>                      <td><c:out value="${book.id}" /></td>                      <td><c:out value="${book.title}" /></td>                      <td><c:out value="${book.author}" /></td>                      <td><c:out value="${book.price}" /></td>                      <td>                          <a href="/edit?id=<c:out value='${book.id}' />">Edit</a>                          &nbsp;&nbsp;&nbsp;&nbsp;                          <a href="/delete?id=<c:out value='${book.id}' />">Delete</a>                      </td>                  </tr>              </c:forEach>          </table>      </div>  </body>  </html> |

In this JSP page, we use JSTL to display records of the table **book** from database. The **listBook** object will be passed from a servlet which we will create later.

On running, this page looks something like this:



As you can see, on this page we have two hyperlinks at the top menu for creating a new book (**Add New Book**) and showing all books (**List All Books**). In addition, for each individual book there are two links for editing (**Edit**) and deleting (**Delete**).

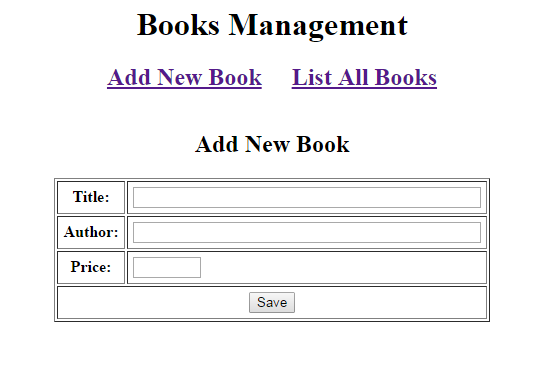
## 6. Writing Book Form JSP Page

Next, we create a JSP page for creating a new book called **BookForm.jsp**. Here’s its full source code:

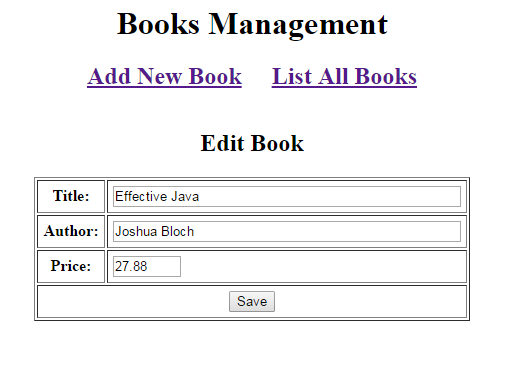
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72 | <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <html>  <head>      <title>Books Store Application</title>  </head>  <body>      <center>          <h1>Books Management</h1>          <h2>              <a href="/new">Add New Book</a>              &nbsp;&nbsp;&nbsp;              <a href="/list">List All Books</a>            </h2>      </center>      <div align="center">          <c:if test="${book != null}">              <form action="update" method="post">          </c:if>          <c:if test="${book == null}">              <form action="insert" method="post">          </c:if>          <table border="1" cellpadding="5">              <caption>                  <h2>                      <c:if test="${book != null}">                          Edit Book                      </c:if>                      <c:if test="${book == null}">                          Add New Book                      </c:if>                  </h2>              </caption>                  <c:if test="${book != null}">                      <input type="hidden" name="id" value="<c:out value='${book.id}' />" />                  </c:if>              <tr>                  <th>Title: </th>                  <td>                      <input type="text" name="title" size="45"                              value="<c:out value='${book.title}' />"                          />                  </td>              </tr>              <tr>                  <th>Author: </th>                  <td>                      <input type="text" name="author" size="45"                              value="<c:out value='${book.author}' />"                      />                  </td>              </tr>              <tr>                  <th>Price: </th>                  <td>                      <input type="text" name="price" size="5"                              value="<c:out value='${book.price}' />"                      />                  </td>              </tr>              <tr>                  <td colspan="2" align="center">                      <input type="submit" value="Save" />                  </td>              </tr>          </table>          </form>      </div>  </body>  </html> |

This page will be served for both creating a new and editing an existing book. In editing mode, the servlet will pass a Book object to the request and we use the JSTL’s **<c:if>**tag to determine whether this object is available or not. If available (not null) the form is in editing mode, otherwise it is in creating mode.

On running, this page shows new form like this:



And on editing mode:



We’ll see how to connect the DAO class with the JSP pages based on user’s requests in the next section: creating the servlet class.

## 7. Coding Controller Servlet Class

Now, the most difficult but interesting part is implement a Java Servlet that acts as a page controller to handle all requests from the client. Let’s look at the code first:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123 | package net.codejava.javaee.bookstore;    import java.io.IOException;  import java.sql.SQLException;  import java.util.List;    import javax.servlet.RequestDispatcher;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;    /\*\*   \* ControllerServlet.java   \* This servlet acts as a page controller for the application, handling all   \* requests from the user.   \* @author www.codejava.net   \*/  public class ControllerServlet extends HttpServlet {      private static final long serialVersionUID = 1L;      private BookDAO bookDAO;        public void init() {          String jdbcURL = getServletContext().getInitParameter("jdbcURL");          String jdbcUsername = getServletContext().getInitParameter("jdbcUsername");          String jdbcPassword = getServletContext().getInitParameter("jdbcPassword");            bookDAO = new BookDAO(jdbcURL, jdbcUsername, jdbcPassword);        }        protected void doPost(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {          doGet(request, response);      }        protected void doGet(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {          String action = request.getServletPath();            try {              switch (action) {              case "/new":                  showNewForm(request, response);                  break;              case "/insert":                  insertBook(request, response);                  break;              case "/delete":                  deleteBook(request, response);                  break;              case "/edit":                  showEditForm(request, response);                  break;              case "/update":                  updateBook(request, response);                  break;              default:                  listBook(request, response);                  break;              }          } catch (SQLException ex) {              throw new ServletException(ex);          }      }        private void listBook(HttpServletRequest request, HttpServletResponse response)              throws SQLException, IOException, ServletException {          List<Book> listBook = bookDAO.listAllBooks();          request.setAttribute("listBook", listBook);          RequestDispatcher dispatcher = request.getRequestDispatcher("BookList.jsp");          dispatcher.forward(request, response);      }        private void showNewForm(HttpServletRequest request, HttpServletResponse response)              throws ServletException, IOException {          RequestDispatcher dispatcher = request.getRequestDispatcher("BookForm.jsp");          dispatcher.forward(request, response);      }        private void showEditForm(HttpServletRequest request, HttpServletResponse response)              throws SQLException, ServletException, IOException {          int id = Integer.parseInt(request.getParameter("id"));          Book existingBook = bookDAO.getBook(id);          RequestDispatcher dispatcher = request.getRequestDispatcher("BookForm.jsp");          request.setAttribute("book", existingBook);          dispatcher.forward(request, response);        }        private void insertBook(HttpServletRequest request, HttpServletResponse response)              throws SQLException, IOException {          String title = request.getParameter("title");          String author = request.getParameter("author");          float price = Float.parseFloat(request.getParameter("price"));            Book newBook = new Book(title, author, price);          bookDAO.insertBook(newBook);          response.sendRedirect("list");      }        private void updateBook(HttpServletRequest request, HttpServletResponse response)              throws SQLException, IOException {          int id = Integer.parseInt(request.getParameter("id"));          String title = request.getParameter("title");          String author = request.getParameter("author");          float price = Float.parseFloat(request.getParameter("price"));            Book book = new Book(id, title, author, price);          bookDAO.updateBook(book);          response.sendRedirect("list");      }        private void deleteBook(HttpServletRequest request, HttpServletResponse response)              throws SQLException, IOException {          int id = Integer.parseInt(request.getParameter("id"));            Book book = new Book(id);          bookDAO.deleteBook(book);          response.sendRedirect("list");        }  } |

First, look at the **init()** method which instantiates an instance of the **BookDAO**class when the servlet is instantiated for the first time. The JDBC connection information will be read from Servlet’s context parameters. This method is invoked only one time during life cycle of the servlet so it’s reasonable to put the DAO instantiation code here:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public void init() {      String jdbcURL = getServletContext().getInitParameter("jdbcURL");      String jdbcUsername = getServletContext().getInitParameter("jdbcUsername");      String jdbcPassword = getServletContext().getInitParameter("jdbcPassword");        bookDAO = new BookDAO(jdbcURL, jdbcUsername, jdbcPassword);    } |

Next, we can see this servlet handles both **GET** and **POST** requests as the **doPost()**method invokes the **doGet()**which handles all the request:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | protected void doGet(HttpServletRequest request, HttpServletResponse response)          throws ServletException, IOException {      String action = request.getServletPath();        try {          switch (action) {          case "/new":              showNewForm(request, response);              break;          case "/insert":              insertBook(request, response);              break;          case "/delete":              deleteBook(request, response);              break;          case "/edit":              showEditForm(request, response);              break;          case "/update":              updateBook(request, response);              break;          default:              listBook(request, response);              break;          }      } catch (SQLException ex) {          throw new ServletException(ex);      }  } |

Based on the request URL (starts with /edit, /list, /new, etc) the servlet calls the corresponding methods. Here we examine one method for example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | private void listBook(HttpServletRequest request, HttpServletResponse response)          throws SQLException, IOException, ServletException {      List<Book> listBook = bookDAO.listAllBooks();      request.setAttribute("listBook", listBook);      RequestDispatcher dispatcher = request.getRequestDispatcher("BookList.jsp");      dispatcher.forward(request, response);  } |

This method uses the DAO class to retrieve all books from the database, and then forward to the **BookList.jsp** page for displaying the result. Similar logic is implemented for the rest methods.

I recommend you to read [this famous Servlet and JSP book](https://amzn.to/2HySkSB) to master Java servlet and JSP.

## 8. Configuring Web.xml

To make the **ControllerServlet**intercepts all requests, we have to configure its mapping in the web deployment descriptor **web.xml**file. Open the web.xml file under **WebContent\WEB-INF**directory and update it with the following code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xmlns="http://xmlns.jcp.org/xml/ns/javaee"      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee          http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"      id="WebApp\_ID" version="3.1">      <display-name>Books Management Web Application</display-name>        <context-param>          <param-name>jdbcURL</param-name>          <param-value>jdbc:mysql://localhost:3306/bookstore</param-value>      </context-param>        <context-param>          <param-name>jdbcUsername</param-name>          <param-value>root</param-value>      </context-param>        <context-param>          <param-name>jdbcPassword</param-name>          <param-value>P@ssw0rd</param-value>      </context-param>        <servlet>          <servlet-name>ControllerServlet</servlet-name>          <servlet-class>net.codejava.javaee.bookstore.ControllerServlet</servlet-class>      </servlet>        <servlet-mapping>          <servlet-name>ControllerServlet</servlet-name>          <url-pattern>/</url-pattern>      </servlet-mapping>        <error-page>          <exception-type>java.lang.Exception</exception-type>          <location>/Error.jsp</location>      </error-page>  </web-app> |

 As you can see, the **<context-param>** elements specify JDBC connection information (URL, username and password) for the DAO class.

The **<servlet>** and **<servlet-mapping>** elements declare and specify URL mapping for the **ControllerServlet**class. The URL pattern **/**means this is the default servlet to handle all requests.

The **<error>** page elements specify error handling page for all kind of exceptions (**java.lang.Exception**) which may occur during the life of the application.

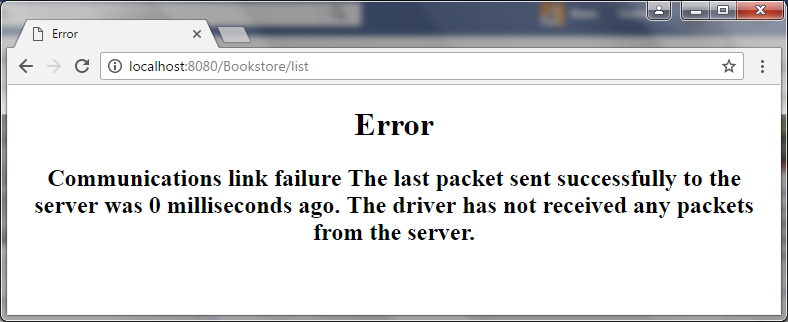
For details about error handling in Java web application, read [this tutorial](https://www.codejava.net/java-ee/servlet/how-to-handle-error-in-web-xml-for-java-web-applications).

## 9. Writing Error JSP page

Here’s the code of the **Error.jsp**page which simply shows the exception message:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <%@ page language="java" contentType="text/html; charset=UTF-8"      pageEncoding="UTF-8" isErrorPage="true" %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"      "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <title>Error</title>  </head>  <body>      <center>          <h1>Error</h1>          <h2><%=exception.getMessage() %><br/> </h2>      </center>  </body>  </html> |

It looks something like this when an error occurs:



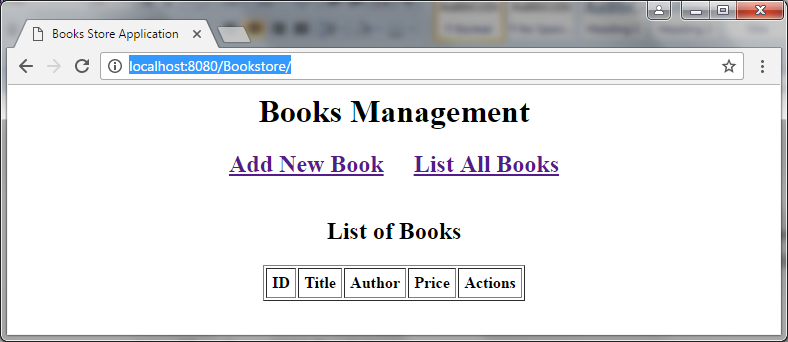
## 10. Deploying and Testing the Application

So far we have completed the code of the project. It’s time to deploy and test the application to see how it works. Follow [this tutorial](https://www.codejava.net/servers/tomcat/how-to-add-tomcat-server-in-eclipse-ide) in case you don’t know how to add Apache Tomcat server in Eclipse.

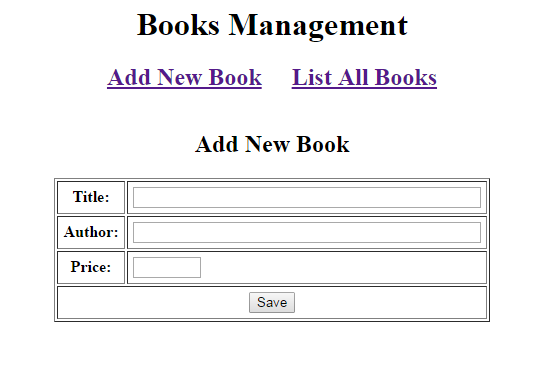
Type the following URL in your web browser to access the Bookstore application:

***http://localhost:8080/Bookstore***

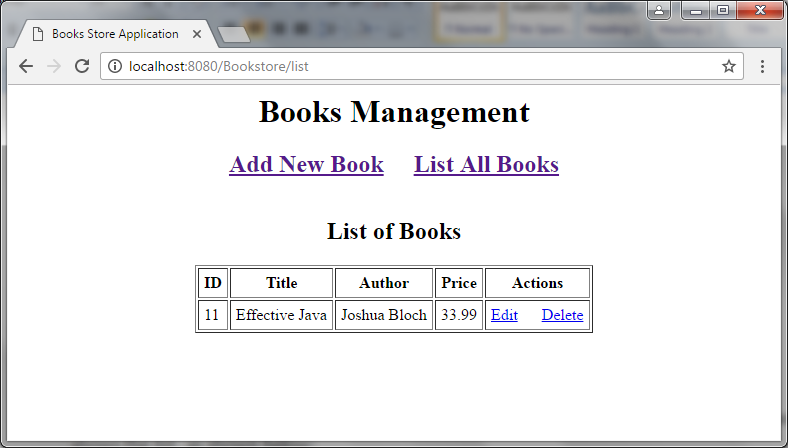
First time, the list is empty because there hasn’t any books yet:



Click on the hyperlink **Add New Book** to begin adding a new book:



Enter book information (title, author and price) and click **Save**. The application saves the book and shows the list, as shown below:



In this list, you can click on the **Edit** and **Delete** hyperlinks to edit and delete a specific book.

That’s how a simple Java web application with Serlvet, JSP, JDBC and MySQL is built. We hope you find this tutorial helpful and you can download the whole project under the **Attachments** section below.

If you want to have a full video training, I recommend you to take my course on Udemy [Java Servlet, JSP and Hibernate: Build a Complete Website](https://www.udemy.com/java-servlet-jsp-and-hibernate-build-a-complete-website/?couponCode=CJA102).

### 

### Related Tutorials:

* [Java Servlet and JSP for beginners](https://www.codejava.net/coding/java-servlet-and-jsp-hello-world-tutorial-with-eclipse-maven-and-apache-tomcat)
* [JDBC Create, Retrieve, Update and Delete (CRUD) Tutorial](https://www.codejava.net/java-se/jdbc/jdbc-tutorial-sql-insert-select-update-and-delete-examples)
* [How to list database records in JSP](https://www.codejava.net/java-ee/jsp/how-to-list-records-in-a-database-table-using-jsp-and-jstl)
* [Handling Error for Java web applications](https://www.codejava.net/java-ee/servlet/how-to-handle-error-in-web-xml-for-java-web-applications)

### 

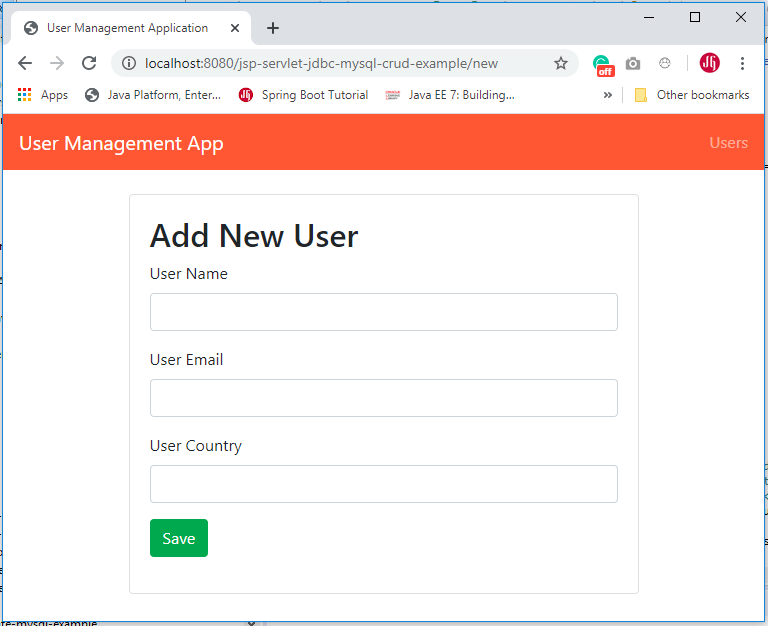
### Other Java Servlet Tutorials:

* [Java Servlet Quick Start for beginners (XML)](https://www.codejava.net/java-ee/servlet/java-servlet-quick-start-tomcat-xml-config)
* [Java Servlet for beginners (annotations)](https://www.codejava.net/java-ee/servlet/quick-start-guide-for-java-servlet-annotations)
* [Handling HTML form data with Java Servlet](https://www.codejava.net/java-ee/servlet/handling-html-form-data-with-java-servlet)
* [Java File Download Servlet Example](https://www.codejava.net/java-ee/servlet/java-servlet-download-file-example)
* [Upload file to servlet without using HTML form](https://www.codejava.net/java-ee/servlet/upload-file-to-servlet-without-using-html-form)
* [How to use Cookies in Java web application](https://www.codejava.net/java-ee/servlet/how-to-use-cookies-in-java-web-application)
* [How to use Session in Java web application](https://www.codejava.net/java-ee/servlet/how-to-use-session-in-java-web-application)

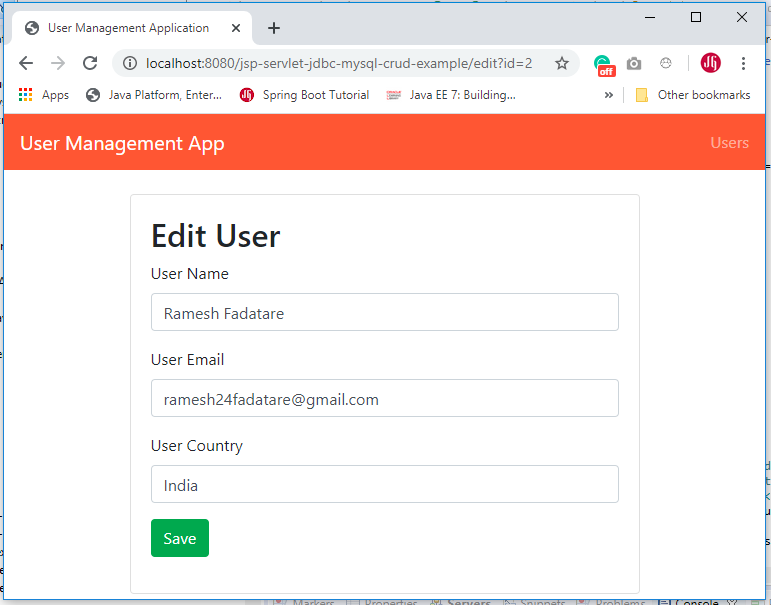
### About the Author:

https://www.codejava.net/images/NamAuthor.png[Nam Ha Minh](https://www.codejava.net/nam-ha-minh) is certified Java programmer (SCJP and SCWCD). He started programming with Java in the time of Java 1.4 and has been falling in love with Java since then. Make friend with him on [Facebook](https://www.facebook.com/namjavaprogrammer) and watch [his Java videos](https://www.youtube.com/codejava?utm_source=codejava&utm_campaign=aboutnamhm) you YouTube.

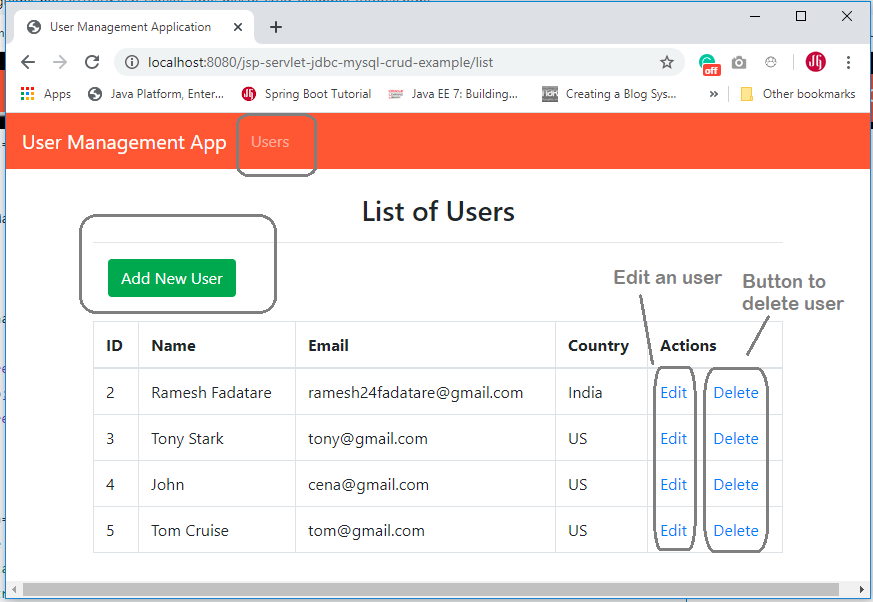
### Create a new User

**[](https://1.bp.blogspot.com/-_mI88FVdmeI/XiRVGjPiotI/AAAAAAAAHYQ/rsZ4iiKtIMAPzrglVrgMtIPPMTx0YRUBACLcBGAsYHQ/s1600/add-user.PNG)**

### Edit a User

**[](https://1.bp.blogspot.com/-ILrXYGx4lwA/XiRVLl18yeI/AAAAAAAAHYU/NFa7YBQzcvkUhNNegRzOe7bj6E4n8_I4QCLcBGAsYHQ/s1600/edit-user.PNG)**

### List of all Users

**[](https://1.bp.blogspot.com/-vv_90lCrWEs/XiRVSKKrtOI/AAAAAAAAHYg/Me65pJye7TIfkvuUb5n3K00ug_N5zh5tACLcBGAsYHQ/s1600/list-users.PNG)**

## GitHub Repository

The source code this tutorial (User Management) is available on my GitHub repository at [**https://github.com/RameshMF/jsp-servlet-jdbc-mysql-crud-tutorial**](https://github.com/RameshMF/jsp-servlet-jdbc-mysql-crud-tutorial).

Check out [**Build Todo App using JSP, Servlet, JDBC, and MySQL**](https://www.javaguides.net/2019/10/build-todo-app-using-jsp-servlet-jdbc-and-mysql.html).

## Servlet + JSP + JDBC + MySQL Examples

* [**Servlet + JSP + JDBC + MySQL Example**](https://www.javaguides.net/2019/03/servlet-jsp-jdbc-mysql-example.html)
* [**Registration Form using JSP + Servlet + JDBC + Mysql Example**](https://www.javaguides.net/2019/03/registration-form-using-jsp-servlet-jdbc-mysql-example.html)
* [**Login Form using JSP + Servlet + JDBC + MySQL Example**](https://www.javaguides.net/2019/03/login-form-using-jsp-servlet-jdbc-mysql-example.html)

<https://gist.github.com/leefsmp/b4c089734852c793cf85>

package com.autodesk.adn.viewanddata;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Collection;

import java.util.HashMap;

import java.util.UUID;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.google.gson.Gson;

/\*\*

 \* Servlet implementation class Models

 \*/

@WebServlet("/models/\*")

public class Models extends HttpServlet {

  private static final long serialVersionUID = 1L;

  class Model{

    public String name;

    public String urn;

    public String id;

    public Model(String \_id, String \_name, String \_urn) {

      name = \_name;

      urn = \_urn;

      id = \_id;

    }

  }

  private Gson \_gson = null;

  private HashMap<String, Model> \_modelsDb = new HashMap<>();

  //Adds some default models to our db

  public Models() {

    super();

    \_gson = new Gson();

    String id1 = UUID.randomUUID().toString();

    String id2 = UUID.randomUUID().toString();

    String id3 = UUID.randomUUID().toString();

    \_modelsDb.put(id1,

      new Model(

       id1,

      "Engine",

      "... base 64 URN ..."));

    \_modelsDb.put(id2,

      new Model(

      id2,

      "Hairdryer",

      "... base 64 URN ..."));

    \_modelsDb.put(id3,

      new Model(

      id3,

      "Plane Engine",

      "... base 64 URN ..."));

  }

  //a utility method to send object

  //as JSON response

  private void sendAsJson(

    HttpServletResponse response,

    Object obj) throws IOException {

    response.setContentType("application/json");

    String res = \_gson.toJson(obj);

    PrintWriter out = response.getWriter();

    out.print(res);

    out.flush();

  }

  // Get models

  // GET/JavaViewer/models/

  // GET/JavaViewer/models/id

  protected void doGet(

      HttpServletRequest request,

      HttpServletResponse response)

          throws ServletException, IOException {

    String pathInfo = request.getPathInfo();

    if(pathInfo == null || pathInfo.equals("/")){

      Collection<Model> models = \_modelsDb.values();

      sendAsJson(response, models);

      return;

    }

    String[] splits = pathInfo.split("/");

    if(splits.length != 2) {

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

    String modelId = splits[1];

    if(!\_modelsDb.containsKey(modelId)) {

      response.sendError(HttpServletResponse.SC\_NOT\_FOUND);

      return;

    }

    sendAsJson(response, \_modelsDb.get(modelId));

    return;

  }

  // Adds new model in DB

  // POST/JavaViewer/models

  protected void doPost(

      HttpServletRequest request,

      HttpServletResponse response)

          throws ServletException, IOException {

    String pathInfo = request.getPathInfo();

    if(pathInfo == null || pathInfo.equals("/")){

      StringBuilder buffer = new StringBuilder();

        BufferedReader reader = request.getReader();

        String line;

        while ((line = reader.readLine()) != null) {

            buffer.append(line);

        }

        String payload = buffer.toString();

        Model model = \_gson.fromJson(payload, Model.class);

        model.id = UUID.randomUUID().toString();

        \_modelsDb.put(model.id, model);

        sendAsJson(response, model);

    }

    else {

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

  }

  // Updates a model in DB

  // PUT/JavaViewer/models/id

  protected void doPut(

      HttpServletRequest request,

      HttpServletResponse response)

          throws IOException, ServletException {

    String pathInfo = request.getPathInfo();

    if(pathInfo == null || pathInfo.equals("/")){

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

    String[] splits = pathInfo.split("/");

    if(splits.length != 2) {

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

    String modelId = splits[1];

    if(!\_modelsDb.containsKey(modelId)) {

      response.sendError(HttpServletResponse.SC\_NOT\_FOUND);

      return;

    }

    StringBuilder buffer = new StringBuilder();

      BufferedReader reader = request.getReader();

      String line;

      while ((line = reader.readLine()) != null) {

          buffer.append(line);

      }

      String payload = buffer.toString();

      Model model = \_gson.fromJson(payload, Model.class);

      model.id = modelId;

      \_modelsDb.put(modelId, model);

      sendAsJson(response, model);

  }

  // Deletes a model in DB

  // DELETE/JavaViewer/models/id

  protected void doDelete(

      HttpServletRequest request,

      HttpServletResponse response)

          throws IOException, ServletException {

    String pathInfo = request.getPathInfo();

    if(pathInfo == null || pathInfo.equals("/")){

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

    String[] splits = pathInfo.split("/");

    if(splits.length != 2) {

      response.sendError(HttpServletResponse.SC\_BAD\_REQUEST);

      return;

    }

    String modelId = splits[1];

    if(!\_modelsDb.containsKey(modelId)) {

      response.sendError(HttpServletResponse.SC\_NOT\_FOUND);

      return;

    }

    Model model = \_modelsDb.get(modelId);

    \_modelsDb.remove(modelId);

    sendAsJson(response, model);

    return;

  }

}

<https://www.sourcecodeexamples.net/2021/09/servlet-crud-rest-api-example-servlet.html>

# Servlet CRUD REST API - Servlet Restful Web Services Example

[*JAVA*](https://www.sourcecodeexamples.net/search/label/Java)[*RESTFUL*](https://www.sourcecodeexamples.net/search/label/RESTFul)[*SERVLET*](https://www.sourcecodeexamples.net/search/label/Servlet)

In this example, we will show you how to create REST APIs using JavaEE Servlet which returns JSON to the client.

We use below HttpServlet class methods to perform CRUD operations:

* **protected void doDelete(HttpServletRequest req, HttpServletResponse resp)**- This method is called by the server (via the service method) to allow a servlet to handle a DELETE request.
* **protected void doGet(HttpServletRequest req, HttpServletResponse resp)**- This method is called by the server (via the service method) to allow a servlet to handle a GET request.
* **protected void doPost(HttpServletRequest req, HttpServletResponse resp)** - This method is called by the server (via the service method) to allow a servlet to handle a POST request.
* **protected void doPut(HttpServletRequest req, HttpServletResponse resp)** - This method is called by the server (via the service method) to allow a servlet to handle a PUT request.

# 1. Create Maven Web Application

Use the below article to create a Maven web project in Eclipse IDE:

[**How to Create a Web Project Using Maven in Eclipse**](https://www.javaguides.net/2018/11/how-to-create-web-project-using-maven-in-eclipse.html)

# 2. Add Maven Dependencies

Open the pom.xml file and add the following dependencies:

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>3.0-alpha-1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>com.google.code.gson</groupId>

<artifactId>gson</artifactId>

<version>2.8.5</version>

</dependency>

<dependency>

<groupId>org.assertj</groupId>

<artifactId>assertj-core</artifactId>

<version>3.11.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

# 3. Create Model - Todo.java

Let's create a *Todo* model class that Servlet returns to the client as JSON:

public class Todo {

private Long id;

private String text;

public Todo() {

}

public Todo(Long id, String text) {

this.id = id;

this.text = text;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

}

# 4. Create Static Data - Todos.java

Let's create a *Todos* class that creates static data:

import java.util.HashMap;

import java.util.Map;

public class Todos {

public static Map<Long, Todo> todos = new HashMap<>();

static {

todos.put(1L, new Todo(1L, "first todo"));

todos.put(2L, new Todo(2L, "second todo"));

}

public static Long nextId() {

return todos.keySet().stream().reduce(Math::max).orElse(0L) + 1L;

}

}

# 5. Create Util Class

Util class to convert Stream to String:

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

import static java.util.stream.Collectors.joining;

public class Util {

public static String readInputStream(InputStream stream) {

return new BufferedReader(new InputStreamReader(stream)).lines().collect(joining("\n"));

}

}

# 6. Create Servlet - CRUD Operations

Let's create a Servlet named TodoServlet that performs CRUD operations:

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import static java.util.stream.Collectors.joining;

public class TodoServlet extends HttpServlet {

private static final Gson GSON = new GsonBuilder().create();

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {

String uri = req.getRequestURI();

Long id = Long.parseLong(uri.substring("/todos/".length()));

String json = GSON.toJson(Todos.todos.get(id));

resp.setStatus(200);

resp.setHeader("Content-Type", "application/json");

resp.getOutputStream().println(json);

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {

String uri = req.getRequestURI();

Long id = Long.parseLong(uri.substring("/todos/".length()));

if (Todos.todos.containsKey(id)) {

resp.setStatus(422);

resp.getOutputStream().println("You cannot created Todo with id " + id + " because it exists!");

}

String json = Util.readInputStream(req.getInputStream());

Todo todo = GSON.fromJson(json, Todo.class);

todo.setId(id);

Todos.todos.put(todo.getId(), todo);

resp.setStatus(201);

resp.setHeader("Content-Type", "application/json");

resp.getOutputStream().println(GSON.toJson(todo));

}

@Override

protected void doPut(HttpServletRequest req, HttpServletResponse resp) throws IOException {

String uri = req.getRequestURI();

Long id = Long.parseLong(uri.substring("/todos/".length()));

if (!Todos.todos.containsKey(id)) {

resp.setStatus(422);

resp.getOutputStream().println("You cannot update Todo with id " + id + " because it doesn't exists!");

}

String json = Util.readInputStream(req.getInputStream());

Todo todo = GSON.fromJson(json, Todo.class);

todo.setId(id);

Todos.todos.put(todo.getId(), todo);

resp.setStatus(200);

resp.setHeader("Content-Type", "application/json");

resp.getOutputStream().println(GSON.toJson(todo));

}

@Override

protected void doDelete(HttpServletRequest req, HttpServletResponse resp) throws IOException {

String uri = req.getRequestURI();

Long id = Long.parseLong(uri.substring("/todos/".length()));

Todo todo = Todos.todos.remove(id);

String json = GSON.toJson(todo);

resp.setStatus(200);

resp.setHeader("Content-Type", "application/json");

resp.getOutputStream().println(json);

}

}

* **protected void doDelete(HttpServletRequest req, HttpServletResponse resp)**- This method is called by the server (via the service method) to allow a servlet to handle a DELETE request.
* **protected void doGet(HttpServletRequest req, HttpServletResponse resp)**- This method is called by the server (via the service method) to allow a servlet to handle a GET request.
* **protected void doPost(HttpServletRequest req, HttpServletResponse resp)** - This method is called by the server (via the service method) to allow a servlet to handle a POST request.
* **protected void doPut(HttpServletRequest req, HttpServletResponse resp)** - This method is called by the server (via the service method) to allow a servlet to handle a PUT request.

# 7. Configure Servlet in Web.xml

Now let's configure Servlet in web.xml file:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>TodoServlet</servlet-name>

<servlet-class>com.sourcecodeexamples.rest.TodoServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>TodoServlet</servlet-name>

<url-pattern>/todos/\*</url-pattern>

</servlet-mapping>

</web-app>

Run in this project in Tomcat server and Test all the APIs using */todos* URL.

<https://gist.github.com/leefsmp/b4c089734852c793cf85>

<https://pasindusri.medium.com/my-first-ever-experience-in-building-servlet-rest-api-78782280842a>

# My first ever experience in building servlet REST API.

Aug 1, 2021

*Java, Maven, Mysql,Tomcat 9, IntelliJ IDEA(Ultimate version), Ubuntu 20.04*

# Creating a maven project.

What is Maven? Maven is a build automation tool that is mostly used in Java projects. Using maven as a build tool in my web servlet project was an amazing experience.



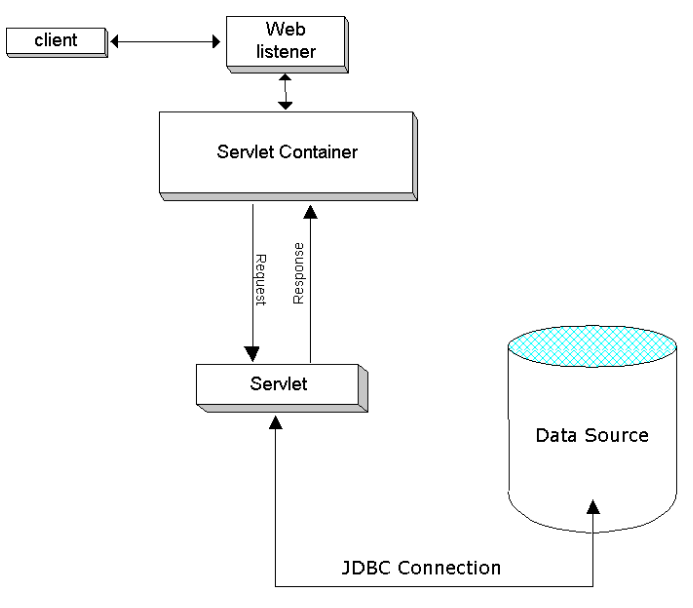
In Maven, a dependency is a JAR, ZIP, or another package that our current project requires to compile, build, test, and/or run. These dependencies are resolved and loaded from the local repository when we perform a build or execute a maven project. The dependencies are gathered in the pom.

The POM provides all required project information as well as plugin configurations for use throughout the build process. The build life cycle is the “when” and “how,” while it is the declarative expression of the “who,” “what,” and “where”.

# Method implementation

A Java servlet does not have a main method. They include methods that you can use to hook into when a specific event occurs (For example - a click on the button, HTTP PUT request on a HttpServlet). During the creation of the servlet, I primarily focused on four methods while implementing. Those are,

1. doPost (POST)
2. doGet (GET)
3. doDelete (DELETE)
4. doUpdate (PUT)



Here is my Github repo regarding my implementation. Just Fork it and refer if you are passionate. <https://github.com/pasindur99/Organizations-backend>

Okay, this will be the end of the first chapter of my experience in creating java servlets. I will be soon again with another chapter on creating java servlets. Thank you for reading!! :)

<https://stackoverflow.com/questions/37157355/using-servlets-to-implement-rest-web-services-in-java>

JAX-RS and Jersey

JAX-RS, currently defined by the [JSR 339](https://jcp.org/en/jsr/detail?id=339), is the standard Java API for creating RESTful web services and it's built the top of the Servlet API.

It's important mention that JAX-RS is an specification. In order to use it, you will need an implementation, such as [Jersey](https://jersey.java.net/), which is the reference implementation.

A few resources that may be useful:

* [JAX-RS 2.0 specification](http://download.oracle.com/otn-pub/jcp/jaxrs-2_0-fr-eval-spec/jsr339-jaxrs-2.0-final-spec.pdf)
* [Jersey documentation](https://jersey.java.net/documentation/latest/user-guide.html)

<https://stackoverflow.com/questions/17888757/what-difference-between-jersey-vs-jax-rs>

JAX-RS is an specification (just a definition) and Jersey is a JAX-RS implementation.