# Spring MVC 5 + Spring Data JPA + Hibernate 5 + JSP + MySQL Tutorial

author: Ramesh Fadatare

hibernate framework spring data jpa tutorial spring mvc tutorial

← Previous Next →

In this tutorial, we will discuss the integration of **Spring MVC 5**, **Spring Data JPA**, Hibernate 5 and MySQL CRUD example. We will demonstrate CRUD(Create, Retrieve, Update, Delete) operations on a Customer entity as well as display list of customers from the MySQL database.

In this tutorial, we will use a Java-based spring configuration to configure **Spring MVC 5**, **Spring Data JPA**, Hibernate 5 and MySQL etc.

**Spring Data JPA** provides CRUD API, so you don't have to write boilerplate code. You just need to create a repository interface and spring will provide

> Before you get started, check out **Spring Data JPA Tutorial** and **Spring MVC Tutorial**

## Spring MVC Tutorials and Articles

- **Spring MVC 5 - Hello World Example**
- **Spring MVC 5 - Sign Up Form Handling Example**
- **Spring MVC JSP Form Tags Tutorial**
- **Spring MVC 5 Form Validation with Annotations Tutorial**
- **Spring MVC 5 + Hibernate 5 + JSP + MySQL CRUD Tutorial**
- **Spring MVC 5 + Spring Data JPA + Hibernate 5 + JSP + MySQL Tutorial**
- **Spring MVC + Spring Boot2 + JSP + JPA + Hibernate 5 + MySQL Example**
- **Spring Boot 2 MVC Web Application Thymeleaf JPA MySQL Example**
- **Spring MVC + Spring Boot2 + JSP + JPA + Hibernate 5 + MySQL Example**
- **Spring Boot 2 MVC Web Application Thymeleaf JPA MySQL Example**
- **Spring Boot 2 - Spring MVC + Thymeleaf Input Form Validation**
- **Spring Boot 2 + Spring MVC + Spring Security + JPA + Thymeleaf + MySQL Tutorial**
- **Authenticating a User with LDAP using Spring Boot and Spring Security**
- **The Spring @Controller and @RestController Annotations with Examples**
- **Spring @RequestBody and @ResponseBody Annotations**
- **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, and @PatchMapping**

**Spring Security**

- **User Registration Module using Spring Boot + Spring MVC + Spring Security + Hibernate 5**

## Video

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

## Tools and technologies used

- Spring MVC - 5.1.0 RELEASE
- Spring Data JPA - 2.10 RELEASE
- Hibernate - 5.2.17.Final
- JDK - 1.8 or later
- Maven - 3.5.1

- JSTL - 1.2.1
- JSP - 2.3.1

Let me list out development steps so that it will be easy to develop and understand Spring MVC application step by step.

## Development Steps

1. Create Maven Web Application
2. Add Dependencies - pom.xml File
3. Project Structure
4. AppInitializer - Register a DispatcherServlet using Java-based Spring configuration
5. PersistenceJPAConfig - Spring Data JPA and Hibernate Configuration using Java-based Spring configuration
6. WebMvcConfig - Spring MVC Bean Configuration using Java-based Spring configuration
7. JPA Entity - Customer.java
8. Spring MVC Controller Class - CustomerController.java
9. Service Layer - CustomerService.java and CustomerServiceImpl.java
10. Spring Data JPA Repository - CustomerRepository.java
11. Custom Exception - ResourceNotFoundException.java
12. JSP Views - customer-form.jsp and list-customers.jsp
13. Serve Static Resources - CSS and JS
14. Build and Run an application
15. Demo

## 1. Create Maven Web Application

1. Use **Guide to Create Maven Web Application** link to create a maven project using a command line.

2. Use **Create Maven Web Application using Eclipse IDE** link to create maven web application using IDE Eclipse.

Once you created maven web application, refer below pom.xml file jar dependencies.

> If you are new to maven then learn maven on **Apache Maven Tutorial**

## 2. Add Dependencies - pom.xml File

```
<project
    xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.ap
    <modelVersion>4.0.0</modelVersion>
    <groupId>net.javaguides.springmvc</groupId>
    <artifactId>springmvc5-springdatajpa2-jsp-mysql-example</artifactI
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>springmvc5-springdatajpa2-jsp-mysql-example Maven Webapp</na
    <url>http://maven.apache.org</url>
    <properties>
        <failOnMissingWebXml>false</failOnMissingWebXml>
        <spring.version>5.1.0.RELEASE</spring.version>
        <hibernate.version>5.2.17.Final</hibernate.version>
        <hibernate.validator>5.4.1.Final</hibernate.validator>
        <c3p0.version>0.9.5.2</c3p0.version>
        <jstl.version>1.2.1</jstl.version>
        <tld.version>1.1.2</tld.version>
        <servlets.version>3.1.0</servlets.version>
        <jsp.version>2.3.1</jsp.version>
```

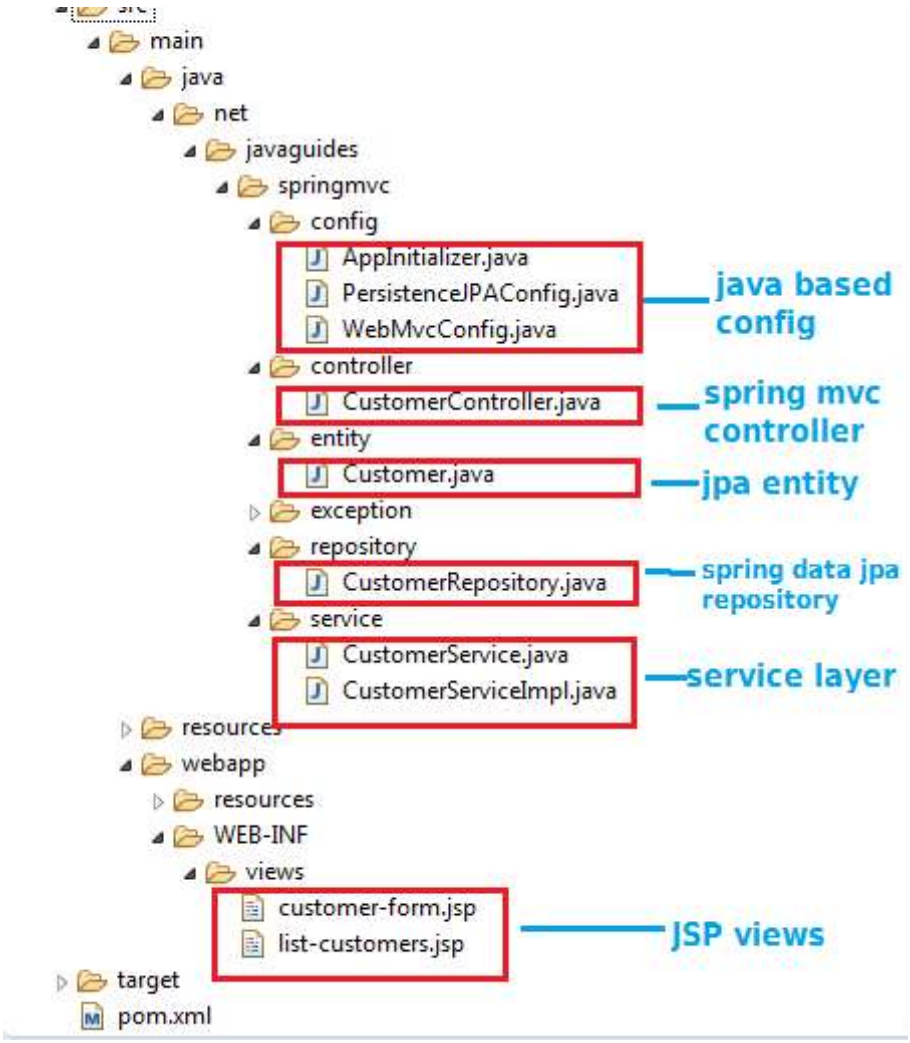
```
<dependencies>
    <!-- Spring MVC Dependency -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <!-- Spring ORM -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate.version}</version>
    </dependency>
    <!-- Hibernate Validator -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>${hibernate.validator}</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework.da
    <dependency>
        <groupId>org.springframework.data</groupId>
        <artifactId>spring-data-jpa</artifactId>
        <version>2.1.0.RELEASE</version>
    </dependency>
    <!-- JSTL Dependency -->
    <dependency>
        <groupId>javax.servlet.jsp.jstl</groupId>
```

```
        <dependency>
            <groupId>taglibs</groupId>
            <artifactId>standard</artifactId>
            <version>${tld.version}</version>
        </dependency>
        <!-- Servlet Dependency -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>${servlets.version}</version>
            <scope>provided</scope>
        </dependency>
        <!-- JSP Dependency -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>${jsp.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!-- logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>1.7.20</version>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.1.7</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
```

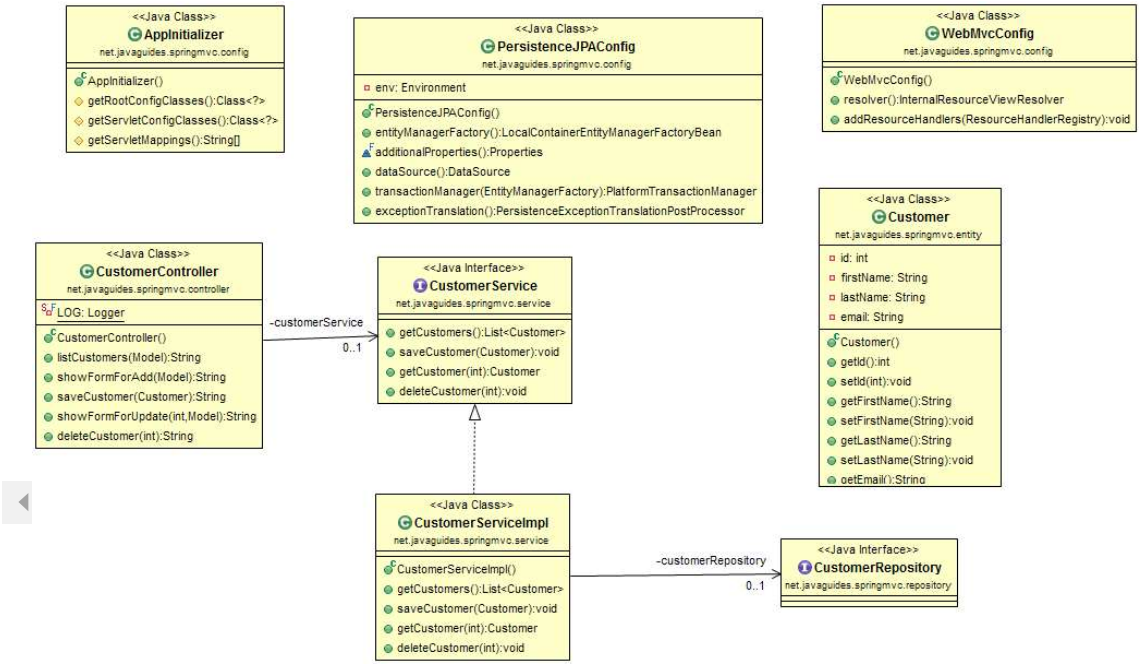
```
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

## 3. Project Structure

Standard project structure for your reference. Refer below project structure throughout this spring MVC application development.




## Class Diagram



In Spring MVC, The `DispatcherServlet` needs to be declared and mapped for processing all requests either using java or web.xmlconfiguration.

In a Servlet 3.0+ environment, you can use `AbstractAnnotationConfigDispatcherServletInitializer` class to register and initialize the `DispatcherServlet` programmatically as follows.

```
package net.javaguides.springmvc.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfi

/**
 * @author Ramesh Fadatare
 */
public class AppInitializer extends AbstractAnnotationConfigDispatcher

    @Override
    protected Class << ? > [] getRootConfigClasses() {
        return new Class[] {
            PersistenceJPAConfig.class
        };
        //return null;
    }

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

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

# 5. PersistenceJPAConfig - Spring Data JPA and Hibernate Configuration using Java-based Spring configuration

## Database Configuration - database.properties

```
# jdbc.X
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/demo?useSSL=false
jdbc.user=root
jdbc.pass=root

# hibernate.X
hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql=false
hibernate.hbm2ddl.auto=create-drop
hibernate.cache.use_second_level_cache=false
hibernate.cache.use_query_cache=false
```

## PersistenceJPAConfig.java

Please note that the `@EnableJpaRepositories` annotation which enables usage of JPA repositories. The `net.javaguides.springmvc.repository` package will be scanned to detect repositories. In the `entityManagerFactory` bean, I determined that `Hibernate` will be used as JPA implementation.

`@EnableTransactionManagement` annotation enables Spring's annotation-driven transaction management capability, similar to the support found in Spring's **tx:*** XML namespace.

```
package net.javaguides.springmvc.config;

import java.util.Properties;
```

```
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.dao.annotation.PersistenceExceptionTranslat
import org.springframework.data.jpa.repository.config.EnableJpaReposit
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryB
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionMar

@Configuration
@EnableTransactionManagement
@PropertySource({
    "classpath:database.properties"
})
@ComponentScan({
    "net.javaguides.springmvc"
})
@EnableJpaRepositories(basePackages = "net.javaguides.springmvc.reposi
public class PersistenceJPAConfig {

    @Autowired
    private Environment env;

    public PersistenceJPAConfig() {
        super();
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory
        final LocalContainerEntityManagerFactoryBean entityManagerFact
        entityManagerFactoryBean.setDataSource(dataSource());
```

```
        });

        final HibernateJpaVendorAdapter vendorAdapter = new HibernateJ
        entityManagerFactoryBean.setJpaVendorAdapter(vendorAdapter);
        entityManagerFactoryBean.setJpaProperties(additionalProperties

        return entityManagerFactoryBean;
    }

    final Properties additionalProperties() {
        final Properties hibernateProperties = new Properties();
        hibernateProperties.setProperty("hibernate.hbm2ddl.auto", env.
        hibernateProperties.setProperty("hibernate.dialect", env.getPr
        hibernateProperties.setProperty("hibernate.cache.use_second_le
        hibernateProperties.setProperty("hibernate.cache.use_query_cac
        // hibernateProperties.setProperty("hibernate.globally_quoted_
        return hibernateProperties;
    }

    @Bean
    public DataSource dataSource() {
        final DriverManagerDataSource dataSource = new DriverManagerDa
        dataSource.setDriverClassName(env.getProperty("jdbc.driverClas
        dataSource.setUrl(env.getProperty("jdbc.url"));
        dataSource.setUsername(env.getProperty("jdbc.user"));
        dataSource.setPassword(env.getProperty("jdbc.pass"));
        return dataSource;
    }

    @Bean
    public PlatformTransactionManager transactionManager(final EntityM
        final JpaTransactionManager transactionManager = new JpaTransa
        transactionManager.setEntityManagerFactory(emf);
        return transactionManager;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor exceptionTrans
        return new PersistenceExceptionTranslationPostProcessor();
```

## 6. WebMvcConfig - Spring MVC Bean Configuration using Java-based Spring configuration

Create an MVCConfig class and annotated with `@Configuration`, `@EnableWebMvc`, and `@ComponentScan` annotations.

```
package net.javaguides.springmvc.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandl
import org.springframework.web.servlet.config.annotation.WebMvcConfigu
import org.springframework.web.servlet.view.InternalResourceViewResolv
import org.springframework.web.servlet.view.JstlView;

/**
 * @author Ramesh Fadatare
 */

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = {
    "net.javaguides.springmvc.controller"
})
public class WebMvcConfig implements WebMvcConfigurer {

    @Bean
    public InternalResourceViewResolver resolver() {
        InternalResourceViewResolver resolver = new InternalResourceVi
        resolver.setViewClass(JstlView.class);
        resolver.setPrefix("/WEB-INF/views/");
```

```
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry)
        registry
            .addResourceHandler("/resources/**")
            .addResourceLocations("/resources/");
    }
}
```

## 7. JPA Entity - Customer.java

```
package net.javaguides.springmvc.entity;

import javax.persistence.Column;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "customer")
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;
```

```
public Customer() {

}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

@Override
public String toString() {
```

```
}
```

## 8. Spring MVC Controller Class - CustomerController.java

Let's create CustomerController Class with CRUD Customer operations.

```
package net.javaguides.springmvc.controller;

import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import net.javaguides.springmvc.entity.Customer;
import net.javaguides.springmvc.exception.ResourceNotFoundException;
import net.javaguides.springmvc.service.CustomerService;

@Controller
@RequestMapping("/customer")
public class CustomerController {

    private static final Logger LOG = LoggerFactory.getLogger(Customer

    @Autowired
    private CustomerService customerService;

    @GetMapping("/list")
    public String listCustomers(Model theModel) {
```

```
        return "list-customers";
    }

    @GetMapping("/showForm")
    public String showFormForAdd(Model theModel) {
        LOG.debug("inside show customer-form handler method");
        Customer theCustomer = new Customer();
        theModel.addAttribute("customer", theCustomer);
        return "customer-form";
    }

    @PostMapping("/saveCustomer")
    public String saveCustomer(@ModelAttribute("customer") Customer th
        customerService.saveCustomer(theCustomer);
        return "redirect:/customer/list";
    }

    @GetMapping("/updateForm")
    public String showFormForUpdate(@RequestParam("customerId") int th
        Model theModel) throws ResourceNotFoundException {
        Customer theCustomer = customerService.getCustomer(theId);
        theModel.addAttribute("customer", theCustomer);
        return "customer-form";
    }

    @GetMapping("/delete")
    public String deleteCustomer(@RequestParam("customerId") int theId
        customerService.deleteCustomer(theId);
        return "redirect:/customer/list";
    }
}
```

# 9. Service Layer - CustomerService.java and CustomerServiceImpl.java

## CustomerService.java

```
package net.javaguides.springmvc.service;

import java.util.List;

import net.javaguides.springmvc.entity.Customer;
import net.javaguides.springmvc.exception.ResourceNotFoundException;

public interface CustomerService {

    public List < Customer > getCustomers();

    public void saveCustomer(Customer theCustomer);

    public Customer getCustomer(int theId) throws ResourceNotFoundExce

    public void deleteCustomer(int theId) throws ResourceNotFoundExcep
}
```

## CustomerServiceImpl.java

```
package net.javaguides.springmvc.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import net.javaguides.springmvc.entity.Customer;
import net.javaguides.springmvc.exception.ResourceNotFoundException;
import net.javaguides.springmvc.repository.CustomerRepository;

@Service
public class CustomerServiceImpl implements CustomerService {

    @Autowired
    private CustomerRepository customerRepository;
```

```
    @Transactional
    public List < Customer > getCustomers() {
        return customerRepository.findAll();
    }

    @Override
    @Transactional
    public void saveCustomer(Customer theCustomer) {
        customerRepository.save(theCustomer);
    }

    @Override
    @Transactional
    public Customer getCustomer(int id) throws ResourceNotFoundExcepti
        return customerRepository.findById(id).orElseThrow(
            () - > new ResourceNotFoundException(id));
    }

    @Override
    @Transactional
    public void deleteCustomer(int theId) {
        customerRepository.deleteById(theId);
    }
}
```

## 10. Spring Data JPA Repository - CustomerRepository.java

```
package net.javaguides.springmvc.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import net.javaguides.springmvc.entity.Customer;

@Repository("customerRepository")
public interface CustomerRepository extends JpaRepository<Customer, Ir
```

## 11. Custom Exception - ResourceNotFoundException.java

```
package net.javaguides.springmvc.exception;

public class ResourceNotFoundException extends Exception {
    private static final long serialVersionUID = 1 L;

    public ResourceNotFoundException(Object resourId) {
        super(resourId != null ? resourId.toString() : null);
    }
}
```

## 12. JSP Views - customer-form.jsp and list-customers.jsp

### customer-form.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Spring MVC 5 - form handling | Java Guides</title>
<link href="<c:url value="/resources/css/bootstrap.min.css" />"
 rel="stylesheet">
<script src="<c:url value="/resources/js/jquery-1.11.1.min.js" />"></s
<script src="<c:url value="/resources/js/bootstrap.min.js" />"></scrip

</head>
<body>
```

```
<h3 class="text-center">Spring MVC 5 + Spring Data JPA 2 + JSP + Hi
 Example - Customer Management</h3>
<div class="panel panel-info">
 <div class="panel-heading">
  <div class="panel-title">Add Customer</div>
 </div>
 <div class="panel-body">
  <form:form action="saveCustomer" cssClass="form-horizontal"
   method="post" modelAttribute="customer">

   <!-- need to associate this data with customer id -->
   <form:hidden path="id" />

   <div class="form-group">
    <label for="firstname" class="col-md-3 control-label">First
     Name</label>
    <div class="col-md-9">
     <form:input path="firstName" cssClass="form-control" />
    </div>
   </div>
   <div class="form-group">
    <label for="lastname" class="col-md-3 control-label">Last
     Name</label>
    <div class="col-md-9">
     <form:input path="lastName" cssClass="form-control" />
    </div>
   </div>

   <div class="form-group">
    <label for="email" class="col-md-3 control-label">Email</label>
    <div class="col-md-9">
     <form:input path="email" cssClass="form-control" />
    </div>
   </div>

   <div class="form-group">
    <!-- Button -->
    <div class="col-md-offset-3 col-md-9">
     <form:button cssClass="btn btn-primary">Submit</form:button>
```

```
     </form:form>
    </div>
   </div>
  </div>
 </div>
</body>
</html>
```

## list-customers.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1
<title>javaguides.net</title>
<link href="<c:url value="/resources/css/bootstrap.min.css" />"
 rel="stylesheet">
<%@ page isELIgnored="false"%>
<script src="<c:url value="/resources/js/jquery-1.11.1.min.js" />"></s
<script src="<c:url value="/resources/js/bootstrap.min.js" />"></scrip
</head>
<body>
 <div class="container">
  <div class="col-md-offset-1 col-md-10">
   <h3 class="text-center">Spring MVC 5 + Spring Data JPA 2 + JSP +
    MySQL Example - Customer Management</h3>
   <hr />

   <input type="button" value="Add Customer"
    onclick="window.location.href='showForm'; return false;"
    class="btn btn-primary" /> <br />
   <br />
   <div class="panel panel-info">
```

```
</div>
<div class="panel-body">
 <table class="table table-striped table-bordered">
  <tr>
   <th>First Name</th>
   <th>Last Name</th>
   <th>Email</th>
   <th>Action</th>
  </tr>

  <!-- loop over and print our customers -->
  <c:forEach var="tempCustomer" items="${customers}">

   <!-- construct an "update" link with customer id -->
   <c:url var="updateLink" value="/customer/updateForm">
    <c:param name="customerId" value="${tempCustomer.id}" />
   </c:url>

   <!-- construct an "delete" link with customer id -->
   <c:url var="deleteLink" value="/customer/delete">
    <c:param name="customerId" value="${tempCustomer.id}" />
   </c:url>

   <tr>
    <td>${tempCustomer.firstName}</td>
    <td>${tempCustomer.lastName}</td>
    <td>${tempCustomer.email}</td>

    <td>
     <!-- display the update link --> <a href="${updateLink}">Upda
     | <a href="${deleteLink}"
     onclick="if (!(confirm('Are you sure you want to delete this
    </td>

   </tr>

  </c:forEach>

 </table>
```

```
  </div>

 </div>
 <div class="footer">
  <p>Footer</p>
 </div>
</body>

</html>
```

## 13. Serve Static Resources - CSS and JS

1. Create a `resource` folder under webapp directory.
2. Create `css` and `js` folders under the `resource` directory.
3. Download and keep `bootstrap.min.css` file under `css` folder
4. download and keep `bootstrap.min.js` and `jquery-1.11.1.min.js` files under the resource directory. Note that bootstrap js is depends on jquery js.

## 14. Build and Run an application

As we are using maven build tool so first, we will need to build this application using following maven command:

```
clean install
```

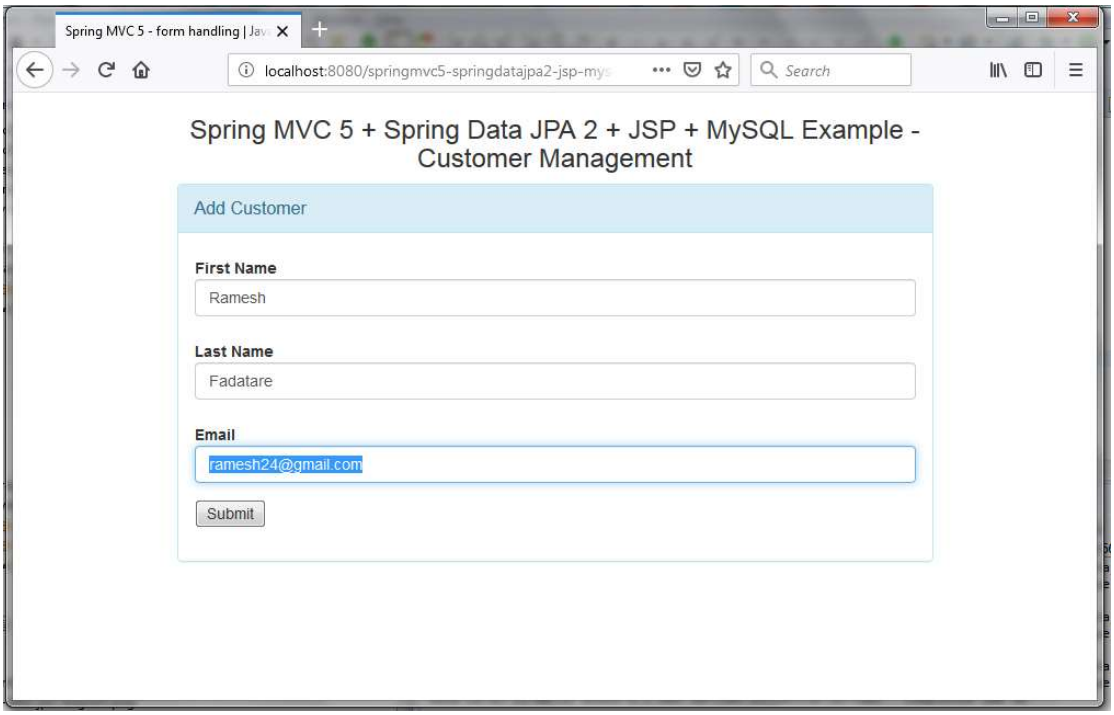
Once build success, then we will run this application on tomcat server 8.5 in IDE or we can also deploy war file on external tomcat webapps folder and run the application.

## 15. Demo

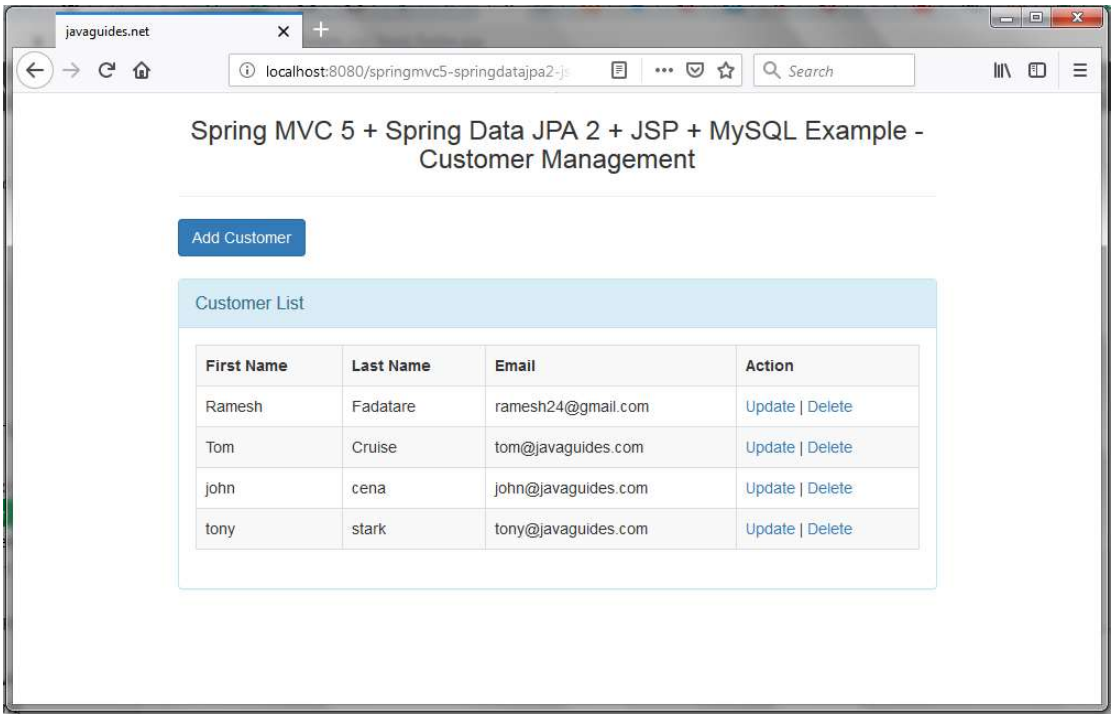
**example/customer/showForm**

On entering the URL, you will see the following page.

### Add Customer page :




### List of customers:



> The source code of this tutorial is available on my **GitHub repository**.

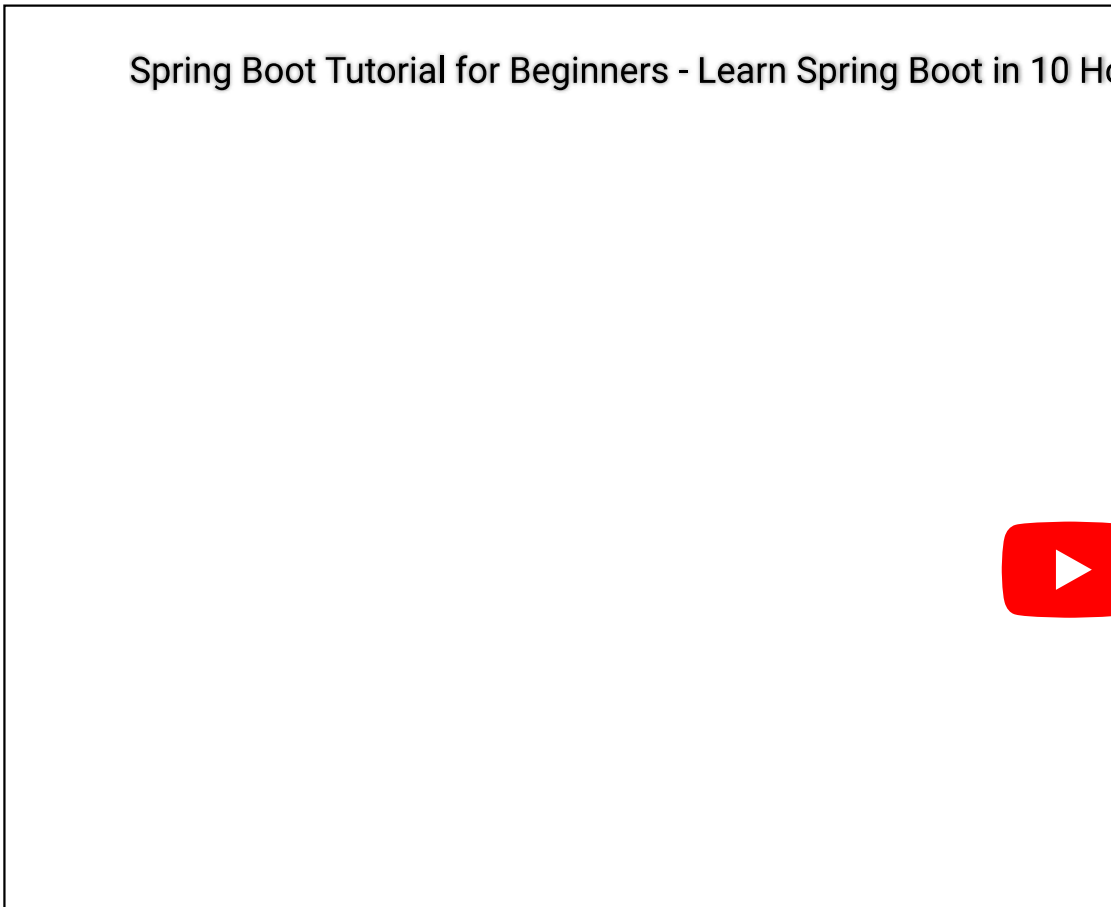
← Previous Next →

hibernate framework spring data jpa tutorial

spring mvc tutorial

# Free Spring Boot Tutorial | Full In-depth Course | Learn Spring Boot in 10 Hours

Watch this course on YouTube at **Spring Boot Tutorial | Fee 10 Hours Full Course**



**manish** 11 July 2020 at 12:30

i tried your project springmvc-5-spring-data-jpa-hibernate-jsp-mysql project i am getting error can u please tell how can i resolve this error
INFO: Initializing Spring root WebApplicationContext
Jul 12, 2020 12:51:35 AM
org.apache.catalina.core.StandardContext listenerStart
SEVERE: Exception sending context initialized event to listener instance of class
[org.springframework.web.context.ContextLoaderListener]
java.lang.NoSuchFieldError: IMPORT_BEAN_NAME_GENERATOR
at
org.springframework.data.repository.config.RepositoryBeanDefinitionRegistrarSupport.registerBeanDefinitions(RepositoryBeanDefinitionRegistrarSupport.java:78)
at
org.springframework.context.annotation.ConfigurationClassBeanDefinitionReader.lambda$loadBeanDefinitionsFromRegistrars$1(ConfigurationClassBeanDefinitionReader.java:364)
at
java.base/java.util.LinkedHashMap.forEach(LinkedHashMap.java:723)
at
org.springframework.context.annotation.ConfigurationClassBeanDefinitionReader.loadBeanDefinitionsFromRegistrars(ConfigurationClassBeanDefinitionReader.java:363)
at
org.springframework.context.annotation.ConfigurationClassBeanDefinitionReader.loadBeanDefinitionsForConfigurationClass(ConfigurationClassBeanDefinitionReader.java:145)
at
org.springframework.context.annotation.ConfigurationClassBeanDefinitionReader.loadBeanDefinitions(ConfigurationClassBeanDefinitionReader.java:117)
at
org.springframework.context.annotation.ConfigurationClassPostProcessor.processConfigBeanDefinitions(ConfigurationClassPostProcessor.java:327)
at
org.springframework.context.annotation.ConfigurationClassPostProcessor.postProcessBeanDefinitionRegistry(ConfigurationClassPostProcessor.java:232)
at
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanDefinitionRegistryPostProcessors(PostProcessorRegistrationDelegate.java:275)
at
org.springframework.context.support.PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors(PostProcessorRegistrationDelegate.java:95)

ava.09+)
at
org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:528)
**REPLY**

To leave a comment, click the button below to sign in with Google

SIGN IN WITH GOOGLE

---


---

## My Udemy Course - Testing Spring Boot Application with JUnit and Mockito


---

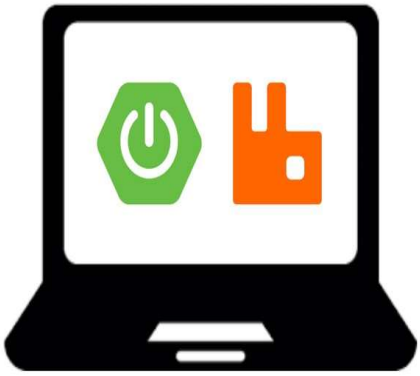
## My Udemy Course - Building Real-Time REST APIs with Spring Boot


---

## My Udemy Course - Master Spring Data JPA with Hibernate

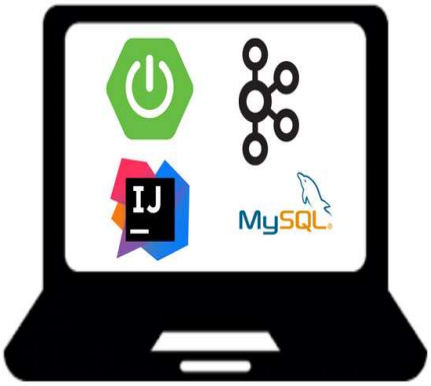

---

## My Udemy Course - Spring Boot RabbitMQ Course - Event-Driven Microservices



---

## My Udemy Course - Spring Boot + Apache Kafka Course




---

## About Me

Hi, I am **Ramesh Fadatare**. I am VMWare Certified Professional for Spring and Spring Boot 2022.

I am founder and author of this blog website **JavaGuides**, a technical blog dedicated to the

with me if you have any questions/queries. Read more about me at **About Me.**

**Top YouTube Channel (75K+ Subscribers):** Check out my YouTube channel for free videos and courses - **Java Guides YouTube Channel**

**My Udemy Courses - https://www.udemy.com/user/ramesh-fadatare/**

Connect with me on **Twitter**, **Facebook**, **LinkedIn**, **GitHub**, and**StackOverflow**

---



---



---



---



---

---



---



Free Courses on YouTube

**Spring Boot Tutorial**

**Java Collections Framework**

**Spring Boot AWS Deployment**

**Spring MVC Tutorial Course**

**5 Spring Boot Projects in 10 Hours**

**Spring Boot Restful Web Services Tutorial**

**Event-Driven Microservices using Spring Boot and Kafka**

**Spring Boot Kafka Real-World Project Tutorial**

Spring Boot Thymeleaf Full Stack

My Udemy Courses

Building Real-Time REST APIs with Spring Boot
Testing Spring Boot Application with JUnit and Mockito
Master Spring Data JPA with Hibernate
Spring Boot + Apache Kafka - The Quickstart Practical Guide
Spring Boot + RabbitMQ (Includes Event-Driven Microservices)
Spring Boot Thymeleaf Real-Time Web Application - Blog App

Connect

YouTube
Twitter
Facebook
GitHub
Linkedin
StackOverflow

Dev Tools

JSON Formatter | Beautifier
Online HTML Editor and Compiler
Base64 Encode Online
Base64 Decode Online
URL Encoder Online
URL Decoder Online