

Sending e-mail with Spring MVC

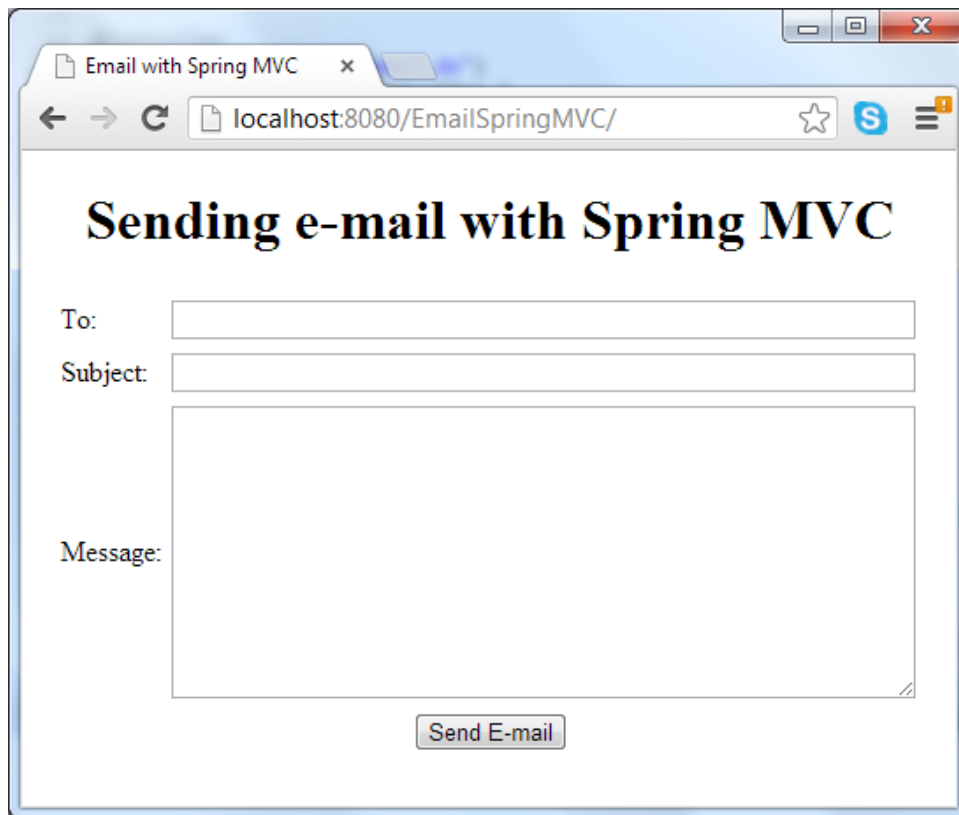
Last Updated on 20 January 2016 | [Print](#) [Email](#)

[Download Aspose for all your file format manipulation needs](#)

Table of contents:

1. Spring framework's support for e-mail
2. Required jar files
3. Creating e-mail sending form
4. Configuring SMTP server settings and Spring MVC
5. Creating Spring MVC controller class
6. Creating result page and error page
7. Run the application
8. Download Eclipse project/WAR file

This tutorial provides a sample spring MVC application that allows user sending an e-mail message by filling a web form. The e-mail form looks like following screenshot:

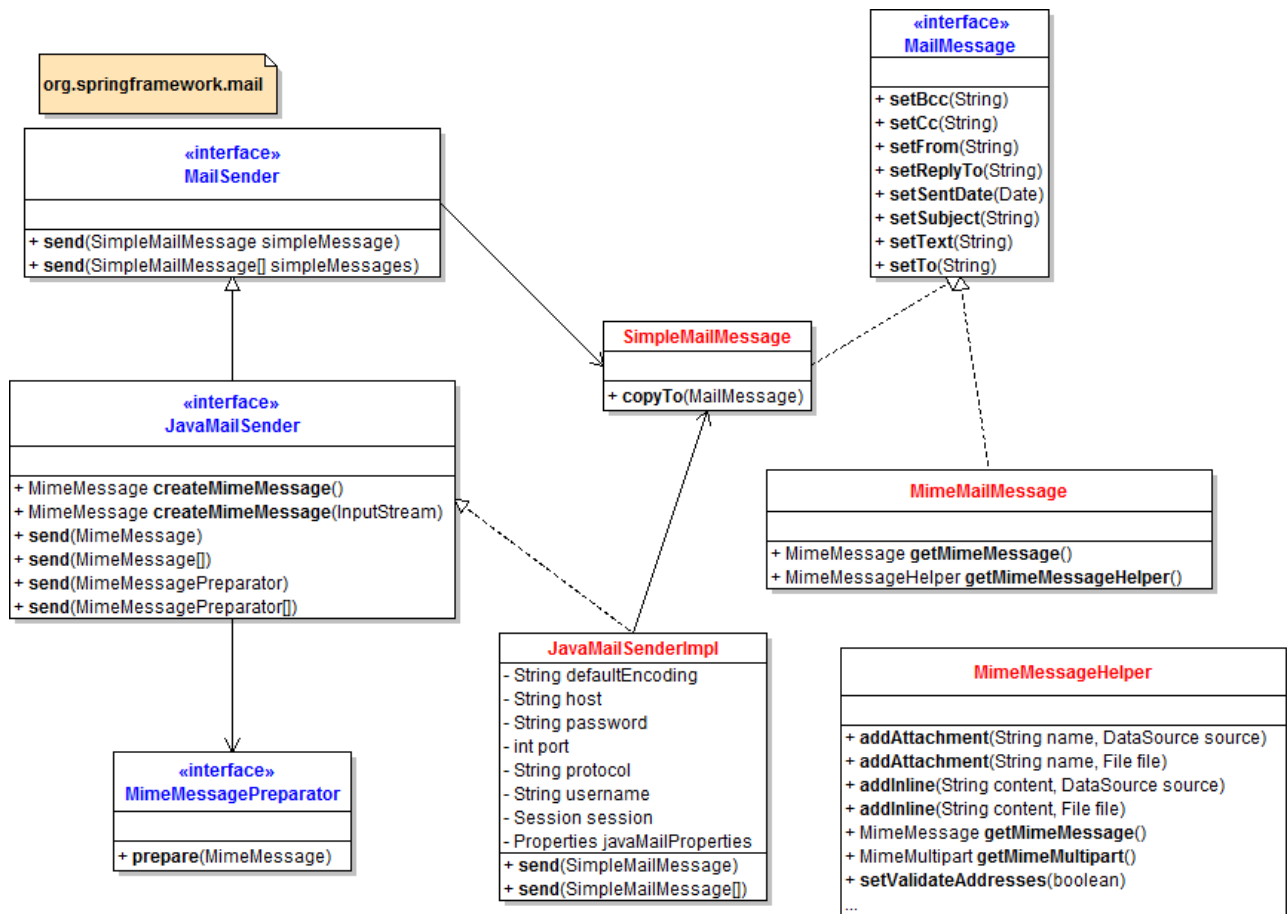


The screenshot shows a web browser window with the title 'Email with Spring MVC'. The address bar displays 'localhost:8080/EmailSpringMVC/'. The main content area has a heading 'Sending e-mail with Spring MVC'. Below the heading, there is a form with three input fields: 'To:', 'Subject:', and 'Message:'. The 'Message:' field is a large text area. At the bottom of the form, there is a button labeled 'Send E-mail'.

In this tutorial, you are supposed to be familiar with Java EE development as well as developing Spring MVC-based applications.

1. Spring framework's support for e-mail

Based on [JavaMail](#), Spring framework provides high-level abstraction API which greatly simplifies e-mail sending process. Let's take a brief look at this API in the following class diagram:



To send e-mail messages, we can use an implementation of interface `MailSender` – the `JavaMailSenderImpl` class which is built upon on JavaMail. It's convenient to configure this implementation as a bean in Spring's context:

```

1  <bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
2      <!-- SMTP settings -->
3      <property name="host" value="SMTP_HOST" />
4      <property name="port" value="SMTP_PORT" />
5      <property name="username" value="USER_NAME" />
6      <property name="password" value="PASSWORD" />
7
8      <property name="javaMailProperties">
9          <!-- additional properties specific to JavaMail -->
10         <props>
11             <prop key="mail.transport.protocol">smtp</prop>
12             <prop key="mail.smtp.auth">true</prop>
13             <prop key="mail.smtp.starttls.enable">true</prop>
14         </props>
15     </property>
16 </bean>
  
```

This bean holds properties for SMTP and JavaMail and can be injected to a business/service class which needs to send an e-mail, for example:

```

1  mailSender.send(email);
  
```



In which `email` is an object of a type that implements `MailMessage` interface, such as `SimpleMailMessage` class. We can construct the `email` object as follows:

```
1 SimpleMailMessage email = new SimpleMailMessage();
2 email.setTo(toAddress);
3 email.setSubject(subject);
4 email.setText(body);
```

That's for a simple mail message (plain text). In case if we want to send HTML e-mail or attach files to the e-mail, we can use `MimeMailMessage` class with the help of `MimeMessagePreparator` class and `MimeMessageHelper` class. For example, sending an e-mail in HTML format with an attachment:

```
1 mailSender.send(new MimeMessagePreparator() {
2     public void prepare(MimeMessage mimeMessage) throws MessagingException {
3         MimeMessageHelper message = new MimeMessageHelper(mimeMessage, true, "UTF-8");
4         message.setFrom(fromEmail);
5         message.setTo(toEmail);
6         message.setSubject("A file for you");
7         message.setText("<b>See the attached</b>", true);
8         message.addAttachment("CoolStuff.doc", new File("CoolStuff.doc"));
9     }
10 });
```

The following table summarizes the interfaces and classes provided in `org.springframework.mail` package:

<code>org.springframework.mail</code>	
MailSender	MailMessage
JavaMailSender	SimpleMailMessage
JavaMailSenderImpl	MimeMailMessage
MimeMessagePreparator	MimeMessageHelper

Click on a link in the table to see API documentation for the corresponding interface/class.

2. Required jar files

The application requires the following jar files copied to its `WEB-INF\lib` directory:

Required jar files	
JavaMail	<code>mail.jar</code>

Spring framework	spring-beans-3.2.0.RELEASE.jar spring-context-3.2.0.RELEASE.jar spring-context-support-3.2.0.RELEASE.jar spring-core-3.2.0.RELEASE.jar spring-expression-3.2.0.RELEASE.jar spring-web-3.2.0.RELEASE.jar spring-webmvc-3.2.0.RELEASE.jar
Apache Commons Logging	commons-logging-1.1.1.jar

NOTE: Click on a hyperlink in the table above to download the corresponding software.

The sample application we are going to build contains the following key files:

- `EmailForm.jsp`: displays an e-mail form.
- `Result.jsp`: shows successful message after the e-mail has been sent.
- `Error.jsp`: shows error message in case of an exception is thrown.
- `SendEmailController.java`: the Spring controller class that takes input from e-mail form, calls Spring's `mailSender` to send the e-mail, and redirects user to either successful page or error page.
- `spring-mvc.xml`: Spring's context configuration file. Here we will configure SMTP server settings and various properties for JavaMail.
- `web.xml`: web deployment descriptor file.

3. Creating e-mail sending form

Create a JSP file called `EmailForm.jsp` with the following HTML code:

```

1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4      "http://www.w3.org/TR/html4/loose.dtd">
5  <html>
6  <head>
7  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8  <title>Email with Spring MVC</title>
9  </head>
10 <body>
11     <center>
12         <h1>Sending e-mail with Spring MVC</h1>
13         <form method="post" action="sendEmail.do">
14             <table border="0" width="80%">
15                 <tr>
16                     <td>To:</td>
17                     <td><input type="text" name="recipient" size="65" /></td>
18                 </tr>
19                 <tr>
20                     <td>Subject:</td>
21                     <td><input type="text" name="subject" size="65" /></td>
22                 </tr>
23                 <tr>
24                     <td>Message:</td>
25                     <td><textarea cols="50" rows="10" name="message"></textarea></td>
26                 </tr>
27                 <tr>
28                     <td colspan="2">
29                         <input type="submit" value="Send E-mail" />
30                     </td>
31                 </tr>
32             </table>
33         </form>
34     </center>
35 </body>
36 </html>

```

This is a simple form with three fields: *To*, *Subject* and *Message* – which are necessary attributes for a simple outgoing e-mail message. On submitting this form, the action named “*sendEmail.do*” will be called, as specified by the form’s `action` attribute. We will implement a Spring controller class for handling this action in the next section.

This book: [Spring in Action](#) helps you learn the latest features, tools, and practices including Spring MVC, REST, Security, Web Flow, and more.

4. Configuring SMTP server settings and Spring MVC

Create a Spring context configuration file called `spring-mvc.xml` with the following XML code:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
6      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
7      http://www.springframework.org/schema/context
8      http://www.springframework.org/schema/context/spring-context-3.0.xsd">
9
10     <context:component-scan base-package="net.codejava.spring" />
11
12     <bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
13         <property name="host" value="smtp.gmail.com" />
14         <property name="port" value="587" />
15         <property name="username" value="youremail" />
16         <property name="password" value="yourpassword" />
17         <property name="javaMailProperties">
18             <props>
19                 <prop key="mail.transport.protocol">smtp</prop>
20                 <prop key="mail.smtp.auth">true</prop>
21                 <prop key="mail.smtp.starttls.enable">true</prop>
22             </props>
23         </property>
24     </bean>
25
26     <bean id="viewResolver"
27         class="org.springframework.web.servlet.view.InternalResourceViewResolver">
28         <property name="prefix" value="/" />
29         <property name="suffix" value=".jsp" />
30     </bean>
31
32     <bean
33         class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
34         <property name="exceptionMappings">
35             <props>
36                 <prop key="java.lang.Exception">Error</prop>
37             </props>
38         </property>
39     </bean>
40 </beans>
```

This configuration is pretty straightforward:

- `<context:component-scan ... />`: tells Spring to scan the package `net.codejava.spring` for initializing components which are annotated by Spring annotations (It's a Spring controller class, in case of this application).
- Bean `mailSender`: this is the important part because it declares a Spring bean for e-mail implementation – the `JavaMailSenderImpl` class and configures SMTP server settings, which is for a Gmail account in this case. This bean will be injected to a Spring controller class which will be covered in the next section.
- Bean `viewResolver`: maps logical view names to real JSP file names.
- Bean `SimpleMappingExceptionResolver`: maps all exceptions of type `java.lang.Exception` to be handled by `Error.jsp` page.

The web deployment descriptor file (`web.xml`) is configured as follows:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xmlns="http://java.sun.com/xml/ns/javaee"
4      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6          http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
7      id="WebApp_ID" version="3.0">
8      <display-name>EmailSpringMVC</display-name>
9
10     <servlet>
11         <servlet-name>SpringController</servlet-name>
12         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
13         <init-param>
14             <param-name>contextConfigLocation</param-name>
15             <param-value>/WEB-INF/spring-mvc.xml</param-value>
16         </init-param>
17         <load-on-startup>1</load-on-startup>
18     </servlet>
19
20     <servlet-mapping>
21         <servlet-name>SpringController</servlet-name>
22         <url-pattern>*.do</url-pattern>
23     </servlet-mapping>
24
25     <welcome-file-list>
26         <welcome-file>EmailForm.jsp</welcome-file>
27     </welcome-file-list>
28 </web-app>

```

It declares Spring controller servlet with its context configuration file (`/WEB-INF/spring-mvc.xml`). The controller is configured to handle all requests whose URL end with pattern: `*.do`. And the default page when accessing the application is the email form (`EmailForm.jsp`).

This video tutorial: [The Java Spring Tutorial](#) help you discover how to master the Spring framework instantly.

5. Creating Spring MVC controller class

In order to handle submission from the e-mail form, we need to create a Spring controller class as follows:

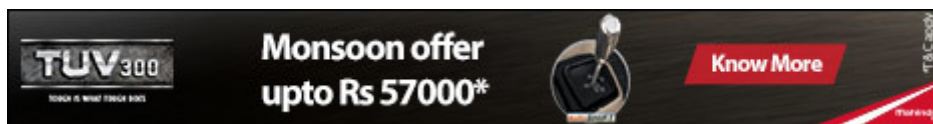
```

1  package net.codejava.spring;
2  import javax.servlet.http.HttpServletRequest;
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.mail.SimpleMailMessage;
5  import org.springframework.mail.javamail.JavaMailSender;
6  import org.springframework.stereotype.Controller;
7  import org.springframework.web.bind.annotation.RequestMapping;
8  import org.springframework.web.bind.annotation.RequestMethod;
9
10 @Controller
11 @RequestMapping("/sendEmail.do")
12 public class SendEmailController {
13
14     @Autowired
15     private JavaMailSender mailSender;
16
17     @RequestMapping(method = RequestMethod.POST)
18     public String doSendEmail(HttpServletRequest request) {
19         // takes input from e-mail form
20         String recipientAddress = request.getParameter("recipient");
21         String subject = request.getParameter("subject");
22         String message = request.getParameter("message");
23
24         // prints debug info
25         System.out.println("To: " + recipientAddress);
26         System.out.println("Subject: " + subject);
27         System.out.println("Message: " + message);
28
29         // creates a simple e-mail object
30         SimpleMailMessage email = new SimpleMailMessage();
31         email.setTo(recipientAddress);
32         email.setSubject(subject);
33         email.setText(message);
34
35         // sends the e-mail
36         mailSender.send(email);
37
38         // forwards to the view named "Result"
39         return "Result";
40     }
41 }

```

This controller class is quite simple. It is declared as a Spring MVC controller by the annotation `@Controller`, and is mapped to the e-mail form's action by the `@RequestMapping` annotation. We inject the `mailSender` bean declared in `spring-mvc.xml` file into this controller through the private field also named `mailSender`. The injection is done automatically by Spring as we use the `@Autowired` annotation.

The method `doSendEmail()` is responsible for capturing input from e-mail form, creating a `SimpleMailMessage` object and sending the e-mail by invoking the `send()` method on the `mailSender` bean. The e-mail is in plain text format. Finally, it returns a view named `"Result"` which causes Spring to use the `viewResolver` to find and load appropriate JSP file (`Result.jsp`).



6. Creating result page and error page

Code the `Result.jsp` file as follows:

```

1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4   "http://www.w3.org/TR/html4/loose.dtd">
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 <title>Send e-mail result</title>
9 </head>
10 <body>
11     <center>
12         <h2>Thank you, your email has been sent.</h2>
13     </center>
14 </body>
15 </html>

```

And code the `Error.jsp` as follows:

```

1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4   "http://www.w3.org/TR/html4/loose.dtd">
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 <title>Error</title>
9 </head>
10 <body>
11     <center>
12         <h2>Sorry, the email was not sent because of the following error:</h2>
13         <h3>${exception.message}</h3>
14     </center>
15 </body>
16 </html>

```

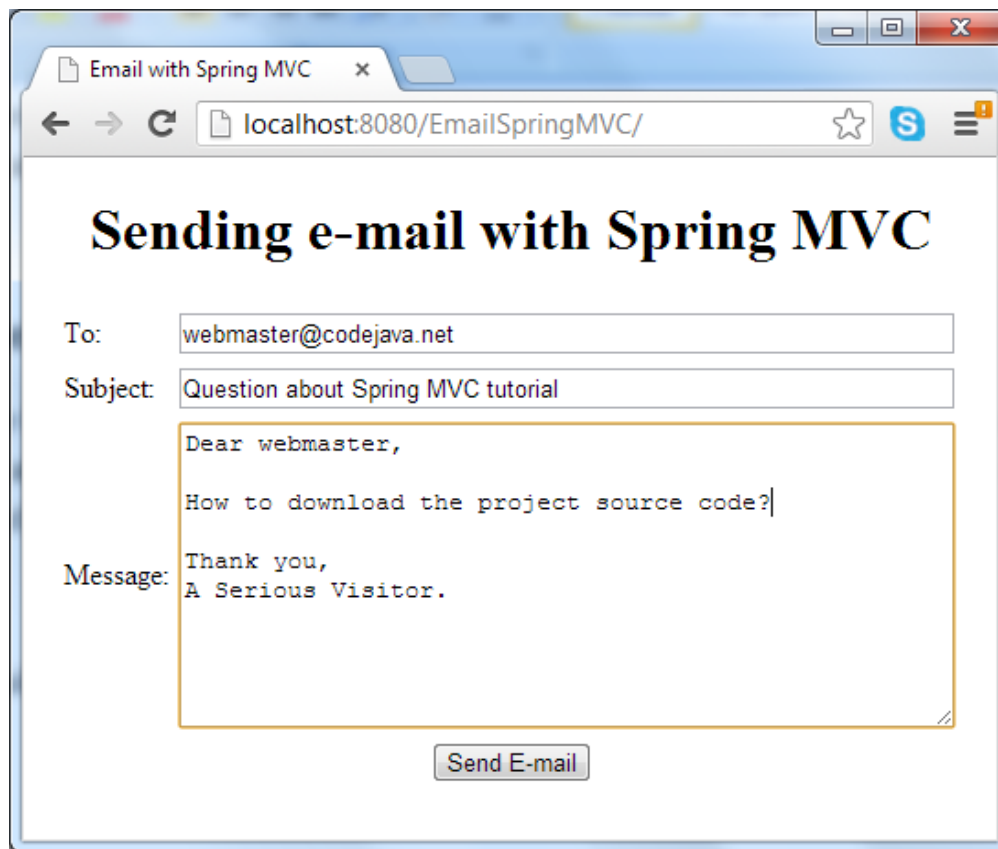
As we can see, the result page simply tells the user that the e-mail has been sent, while the error page displays an error message if any exception thrown during the process of sending e-mail.

7. Run the application

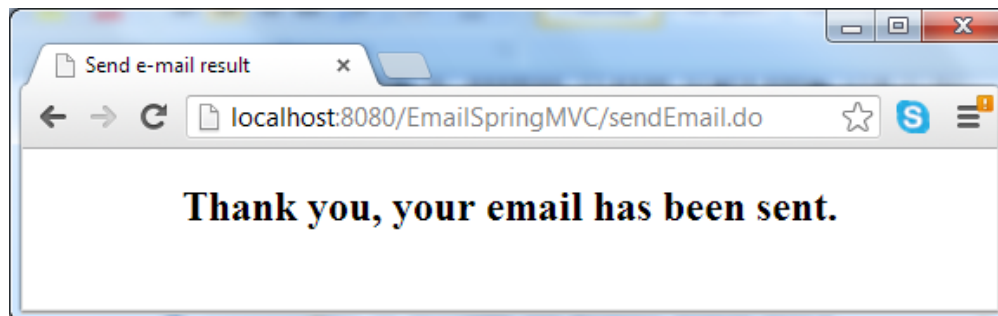
So far we have created all the key pieces of the application. Let's deploy it on a servlet container like Tomcat, and access the application by typing the following URL into browser's address bar (your host name and port number maybe different, depending on server configuration):

<http://localhost:8080/EmailSpringMVC>

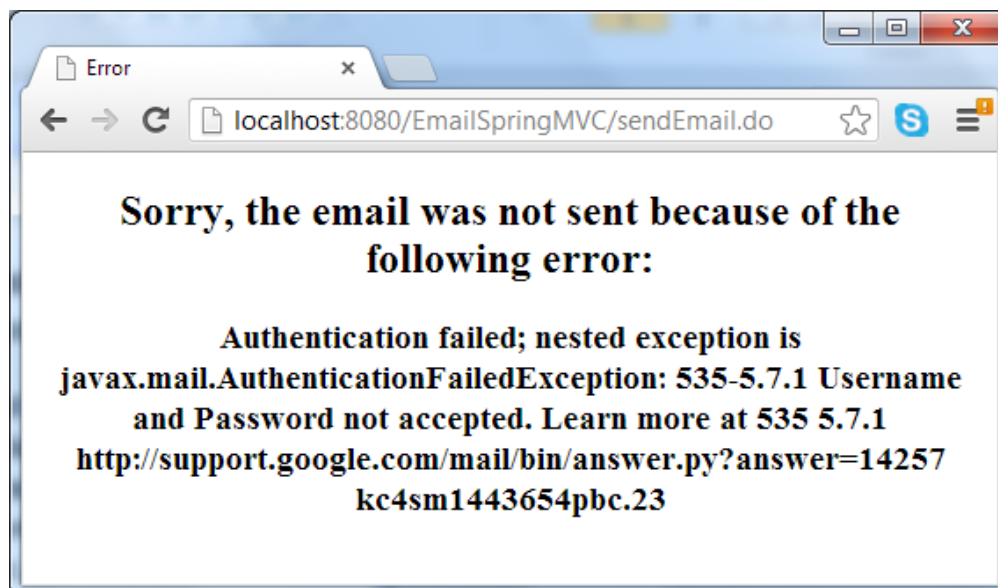
The e-mail form is displayed, type in required information:



Hit **Send E-mail** button, it may take a while for the e-mail to be sent. A successful message comes from the result page in case everything is going well:



In case of error (such as network failure or the SMTP server could not be reached), the error page displays:



You can download the sample application as an Eclipse project or deployable WAR file in the attachment section, and remember to update SMTP settings to match your e-mail account.

You may be also interested in:

- [Spring MVC: Send e-mail with attachments](#)
- [Spring MVC beginner tutorial with Spring Tool Suite IDE](#)
- [Sending e-mail with JSP, Servlet and JavaMail](#)
- [Send e-mail with attachments in Struts2](#)
- [How to start e-mail programming in Java](#)

Share this article:





Free Java Beginner Tutorial Videos (8,212+ guys benefited)

EMAIL ADDRESS:

FIRST NAME:

SEND ME

Attachments:

	EmailSpringMVC.war [Deployable WAR file] 3907 kB
	EmailSpringMVC.zip [Eclipse project] 3908 kB

