



# Chapter 30

## JavaServer™ Faces Web Applications, Part 2

Java How to Program, 9/e



# Chapter 27: JavaServer™ Faces Web Applications, Part 2

Internet & World Wide Web  
How to Program, 5/e

*Note:* This chapter is a copy of Chapter 30 of our book *Java How to Program, 9/e*. For that reason, we simply copied the PowerPoint slides for this chapter and *did not* re-number them



## OBJECTIVES

In this chapter you'll learn:

- To access databases from JSF applications.
- The basic principles and advantages of Ajax technology.
- To use Ajax in a JSF web app.



## 30.1 Introduction

## 30.2 Accessing Databases in Web Apps

30.2.1 Setting Up the Database

30.2.2 @ManagedBean Class AddressBean

30.2.3 index.xhtml Facelets Page

30.2.4 addentry.xhtml Facelets Page

## 30.3 Ajax

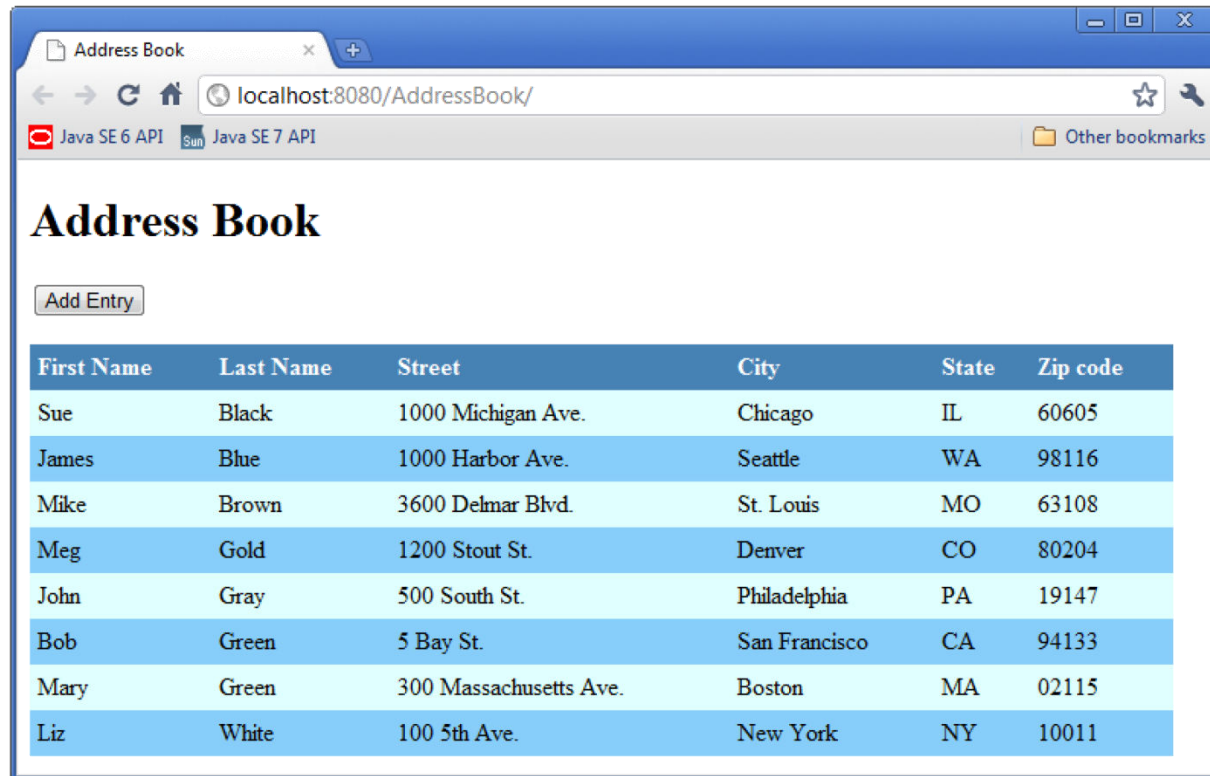
## 30.4 Adding Ajax Functionality to the Validation App

## 30.5 Wrap-Up





a) Table of addresses displayed when the AddressBook app is first requested



First Name	Last Name	Street	City	State	Zip code
Sue	Black	1000 Michigan Ave.	Chicago	IL	60605
James	Blue	1000 Harbor Ave.	Seattle	WA	98116
Mike	Brown	3600 Delmar Blvd.	St. Louis	MO	63108
Meg	Gold	1200 Stout St.	Denver	CO	80204
John	Gray	500 South St.	Philadelphia	PA	19147
Bob	Green	5 Bay St.	San Francisco	CA	94133
Mary	Green	300 Massachusetts Ave.	Boston	MA	02115
Liz	White	100 5th Ave.	New York	NY	10011

**Fig. 30.1** | Sample outputs from the AddressBook app. (Part I of 3.)



b) Form for adding an entry

Address Book: Add Entry

First name:

Last name:

Street:

City:

State:

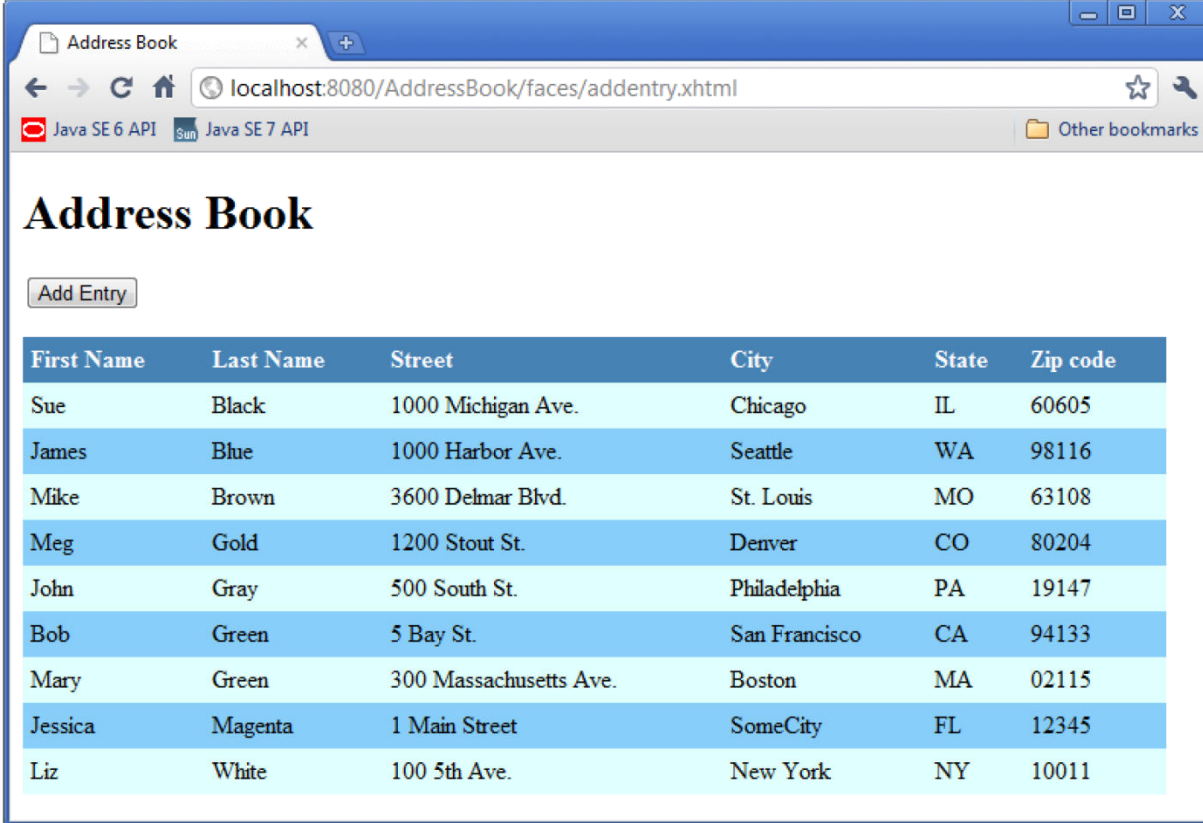
Zipcode:

[Return to Addresses](#)

**Fig. 30.1** | Sample outputs from the AddressBook app. (Part 2 of 3.)

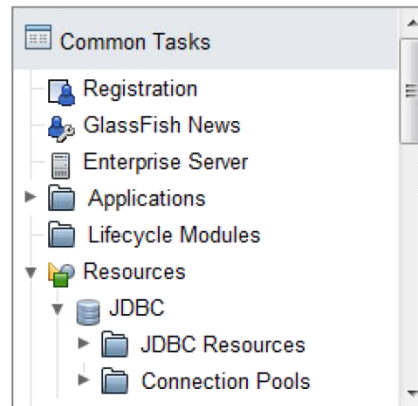


c) Table of addresses updated with the new entry added in Part (b)



First Name	Last Name	Street	City	State	Zip code
Sue	Black	1000 Michigan Ave.	Chicago	IL	60605
James	Blue	1000 Harbor Ave.	Seattle	WA	98116
Mike	Brown	3600 Delmar Blvd.	St. Louis	MO	63108
Meg	Gold	1200 Stout St.	Denver	CO	80204
John	Gray	500 South St.	Philadelphia	PA	19147
Bob	Green	5 Bay St.	San Francisco	CA	94133
Mary	Green	300 Massachusetts Ave.	Boston	MA	02115
Jessica	Magenta	1 Main Street	SomeCity	FL	12345
Liz	White	100 5th Ave.	New York	NY	10011

**Fig. 30.1** | Sample outputs from the AddressBook app. (Part 3 of 3.)



**Fig. 30.2** | **Common Tasks** window in the GlassFish server configuration web page.



---

### New JDBC Connection Pool (Step 1 of 2)

Identify the general settings for the connection pool.

**General Settings**

Name: \*

Resource Type:

Database Vendor:

Select or enter a database vendor

\* Indicates required field

---

**Fig. 30.3** | New JDBC Connection Pool (Step 1 of 2) page.



**Additional Properties (18)**

☒ ☐ | **Add Property** **Delete Properties**

<input type="checkbox"/>	Name	Value
<input type="checkbox"/>	ConnectionAttributes	;create=true
<input type="checkbox"/>	CreateDatabase	
<input type="checkbox"/>	DataSourceName	
<input type="checkbox"/>	DatabaseName	addressbook
<input type="checkbox"/>	Description	
<input type="checkbox"/>	LoginTimeout	0
<input type="checkbox"/>	Password	APP
<input type="checkbox"/>	PortNumber	1527
<input type="checkbox"/>	RetrieveMessageText	true
<input type="checkbox"/>	SecurityMechanism	4
<input type="checkbox"/>	ServerName	localhost
<input type="checkbox"/>	ShutdownDatabase	
<input type="checkbox"/>	Ssl	off
<input type="checkbox"/>	TraceDirectory	
<input type="checkbox"/>	TraceFile	
<input type="checkbox"/>	TraceFileAppend	false
<input type="checkbox"/>	TraceLevel	-1
<input type="checkbox"/>	User	APP

**Fig. 30.4** | New JDBC Connection Pool (Step 2 of 2) page.





## New JDBC Resource

OK Cancel

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: \*

Pool Name:

Use the [JDBC Connection Pools](#) page to create new pools

Description:

Status: ☒ Enabled

### Additional Properties (0)

[Add Property](#) [Delete Properties](#)

Name	Value	Description
No items found.		

**Fig. 30.5** | New JDBC Resource page.



---

```
1  // AddressBean.java
2  // Bean for interacting with the AddressBook database
3  package addressbook;
4
5  import java.sql.Connection;
6  import java.sql.PreparedStatement;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import javax.annotation.Resource;
10 import javax.faces.bean.ManagedBean;
11 import javax.sql.DataSource;
12 import javax.sql.rowset.CachedRowSet;
13
14 @ManagedBean( name="addressBean" )
15 public class AddressBean
16 {
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part I of 10.)



---

```
17    // instance variables that represent one address
18    private String firstName;
19    private String lastName;
20    private String street;
21    private String city;
22    private String state;
23    private String zipcode;
24
25    // allow the server to inject the DataSource
26    @Resource( name="jdbc/addressbook" )
27    DataSource dataSource;
28
29    // get the first name
30    public String getFirstName()
31    {
32        return firstName;
33    } // end method getFirstName
34
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 2 of 10.)



---

```
35    // set the first name
36    public void setFirstName( String firstName )
37    {
38        this.firstName = firstName;
39    } // end method setFirstName
40
41    // get the last name
42    public String getLastName()
43    {
44        return lastName;
45    } // end method getLastName
46
47    // set the last name
48    public void setLastName( String lastName )
49    {
50        this.lastName = lastName;
51    } // end method setLastName
52
53    // get the street
54    public String getStreet()
55    {
56        return street;
57    } // end method getStreet
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 3 of 10.)



---

```
58
59 // set the street
60 public void setStreet( String street )
61 {
62     this.street = street;
63 } // end method setStreet
64
65 // get the city
66 public String getCity()
67 {
68     return city;
69 } // end method getCity
70
71 // set the city
72 public void setCity( String city )
73 {
74     this.city = city;
75 } // end method setCity
76
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 4 of 10.)



---

```
77    // get the state
78    public String getState()
79    {
80        return state;
81    } // end method getState
82
83    // set the state
84    public void setState( String state )
85    {
86        this.state = state;
87    } // end method setState
88
89    // get the zipcode
90    public String getZipcode()
91    {
92        return zipcode;
93    } // end method getZipcode
94
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 5 of 10.)





---

```
95    // set the zipcode
96    public void setZipcode( String zipcode )
97    {
98        this.zipcode = zipcode;
99    } // end method setZipcode
100
101    // return a ResultSet of entries
102    public ResultSet getAddresses() throws SQLException
103    {
104        // check whether dataSource was injected by the server
105        if ( dataSource == null )
106            throw new SQLException( "Unable to obtain DataSource" );
107
108        // obtain a connection from the connection pool
109        Connection connection = dataSource.getConnection();
110
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 6 of 10.)



```
111 // check whether connection was successful
112 if ( connection == null )
113     throw new SQLException( "Unable to connect to DataSource" );
114
115 try
116 {
117     // create a PreparedStatement to insert a new address book entry
118     PreparedStatement getAddresses = connection.prepareStatement(
119         "SELECT FIRSTNAME, LASTNAME, STREET, CITY, STATE, ZIP " +
120         "FROM ADDRESSES ORDER BY LASTNAME, FIRSTNAME" );
121
122     CachedRowSet rowSet = new com.sun.rowset.CachedRowSetImpl();
123     rowSet.populate( getAddresses.executeQuery() );
124     return rowSet;
125 } // end try
126 finally
127 {
128     connection.close(); // return this connection to pool
129 } // end finally
130 } // end method getAddresses
131
```

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 7 of 10.)



---

```
I32    // save a new address book entry
I33    public String save() throws SQLException
I34    {
I35        // check whether dataSource was injected by the server
I36        if ( dataSource == null )
I37            throw new SQLException( "Unable to obtain DataSource" );
I38
I39        // obtain a connection from the connection pool
I40        Connection connection = dataSource.getConnection();
I41
I42        // check whether connection was successful
I43        if ( connection == null )
I44            throw new SQLException( "Unable to connect to DataSource" );
I45
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 8 of 10.)



```
146     try
147     {
148         // create a PreparedStatement to insert a new address book entry
149         PreparedStatement addEntry =
150             connection.prepareStatement( "INSERT INTO ADDRESSES " +
151                 "(FIRSTNAME, LASTNAME, STREET, CITY, STATE, ZIP)" +
152                 "VALUES ( ?, ?, ?, ?, ?, ? )" );
153
154         // specify the PreparedStatement's arguments
155         addEntry.setString( 1, getFirstName() );
156         addEntry.setString( 2, getLastName() );
157         addEntry.setString( 3, getStreet() );
158         addEntry.setString( 4, getCity() );
159         addEntry.setString( 5, getState() );
160         addEntry.setString( 6, getZipcode() );
161
162         addEntry.executeUpdate(); // insert the entry
163         return "index"; // go back to index.xhtml page
164     } // end try
```

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 9 of 10.)



---

```
165     finally
166     {
167         connection.close(); // return this connection to pool
168     } // end finally
169 } // end method save
170 } // end class AddressBean
```

---

**Fig. 30.6** | AddressBean interacts with a database to store and retrieve addresses.  
(Part 10 of 10.)



```
1 <?xml version='1.0' encoding='UTF-8' ?>
2
3 <!-- index.html -->
4 <!-- Displays an h:dataTable of the addresses in the address book -->
5 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7 <html xmlns="http://www.w3.org/1999/xhtml"
8   xmlns:h="http://java.sun.com/jsf/html"
9   xmlns:f="http://java.sun.com/jsf/core">
10 <h:head>
11   <title>Address Book</title>
12   <h:outputStylesheet name="style.css" library="css"/>
13 </h:head>
14 <h:body>
15   <h1>Address Book</h1>
16   <h:form>
17     <p><h:commandButton value="Add Entry" action="addentry"/></p>
18   </h:form>
19   <h:dataTable value="#{addressBean.addresses}" var="address"
20     rowClasses="oddRows,evenRows" headerClass="header"
21     styleClass="table" cellpadding="5" cellspacing="0">
```

**Fig. 30.7** | Displays an `h:dataTable` of the addresses in the address book. (Part 1 of 3.)





```
22      <h:column>
23          <f:facet name="header">First Name</f:facet>
24          #{address.FIRSTNAME}
25      </h:column>
26      <h:column>
27          <f:facet name="header">Last Name</f:facet>
28          #{address.LASTNAME}
29      </h:column>
30      <h:column>
31          <f:facet name="header">Street</f:facet>
32          #{address.STREET}
33      </h:column>
34      <h:column>
35          <f:facet name="header">City</f:facet>
36          #{address.CITY}
37      </h:column>
38      <h:column>
39          <f:facet name="header">State</f:facet>
40          #{address.STATE}
41      </h:column>
```

**Fig. 30.7** | Displays an `h:dataTable` of the addresses in the address book. (Part 2 of 3.)



---

```
42         <h:column>
43             <f:facet name="header">Zip code</f:facet>
44             #{address.ZIP}
45         </h:column>
46     </h:dataTable>
47 </h:body>
48 </html>
```

---

**Fig. 30.7** | Displays an `h:dataTable` of the addresses in the address book. (Part 3 of 3.)



```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- addentry.html -->
4  <!-- Form for adding an entry to an address book -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8    xmlns:h="http://java.sun.com/jsf/html">
9    <h:head>
10      <title>Address Book: Add Entry</title>
11      <h:outputStylesheet name="style.css" library="css"/>
12    </h:head>
13    <h:body>
14      <h1>Address Book: Add Entry</h1>
15      <h:form>
16        <h:panelGrid columns="3">
17          <h:outputText value="First name:"/>
18          <h:inputText id="firstNameInputText" required="true"
19            requiredMessage="Please enter first name"
20            value="#{addressBean.firstName}" maxLength="30"/>
21          <h:message for="firstNameInputText" styleClass="error"/>
22          <h:outputText value="Last name:"/>
```

**Fig. 30.8** | Form for adding an entry to an address book. (Part I of 3.)



```
23      <h:inputText id="lastNameInputText" required="true"
24          requiredMessage="Please enter last name"
25          value="#{addressBean.lastName}" maxLength="30"/>
26      <h:message for="lastNameInputText" styleClass="error"/>
27      <h:outputText value="Street:"/>
28      <h:inputText id="streetInputText" required="true"
29          requiredMessage="Please enter the street address"
30          value="#{addressBean.street}" maxLength="150"/>
31      <h:message for="streetInputText" styleClass="error"/>
32      <h:outputText value="City:"/>
33      <h:inputText id="cityInputText" required="true"
34          requiredMessage="Please enter the city"
35          value="#{addressBean.city}" maxLength="30"/>
36      <h:message for="cityInputText" styleClass="error"/>
37      <h:outputText value="State:"/>
38      <h:inputText id="stateInputText" required="true"
39          requiredMessage="Please enter state"
40          value="#{addressBean.state}" maxLength="2"/>
41      <h:message for="stateInputText" styleClass="error"/>
42      <h:outputText value="Zipcode:"/>
43      <h:inputText id="zipcodeInputText" required="true"
44          requiredMessage="Please enter zipcode"
45          value="#{addressBean.zipcode}" maxLength="5"/>
```

**Fig. 30.8** | Form for adding an entry to an address book. (Part 2 of 3.)

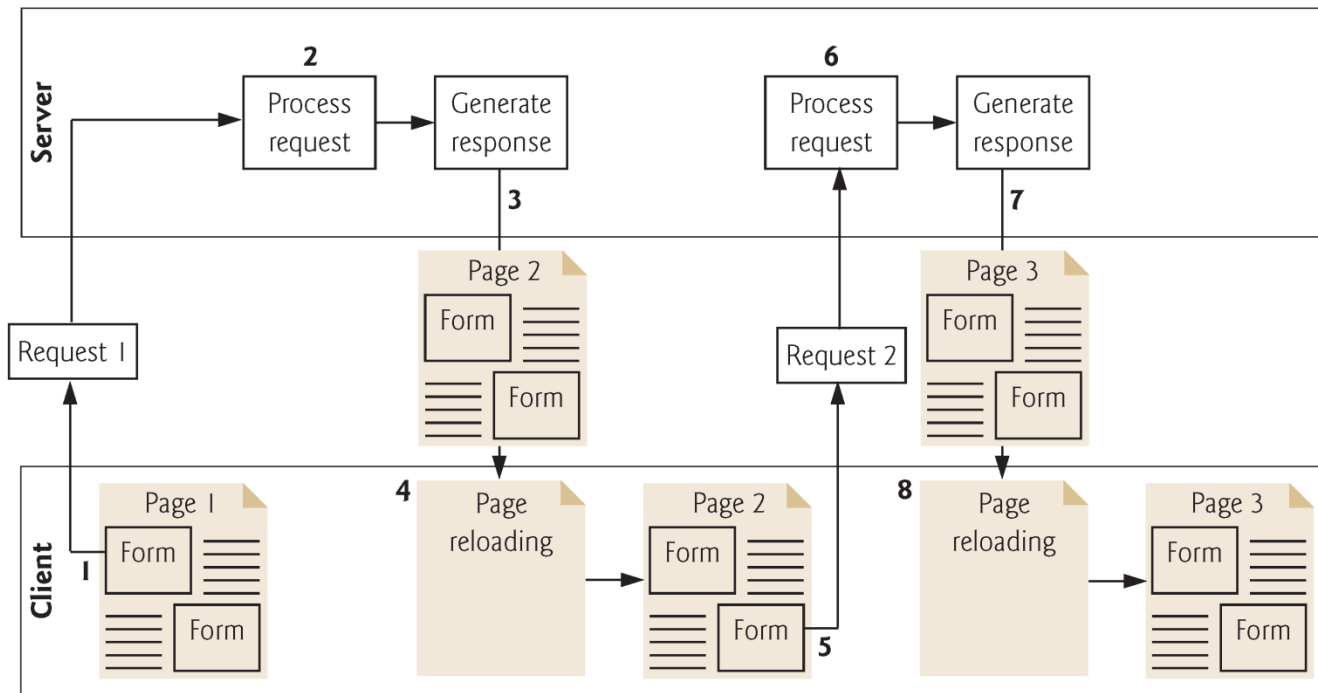


---

```
46         <h:message for="zipcodeInputText" styleClass="error"/>
47     </h:panelGrid>
48     <h:commandButton value="Save Address"
49         action="#{addressBean.save}"/>
50 </h:form>
51 <h:outputLink value="index.xhtml">Return to Addresses</h:outputLink>
52 </h:body>
53 </html>
```

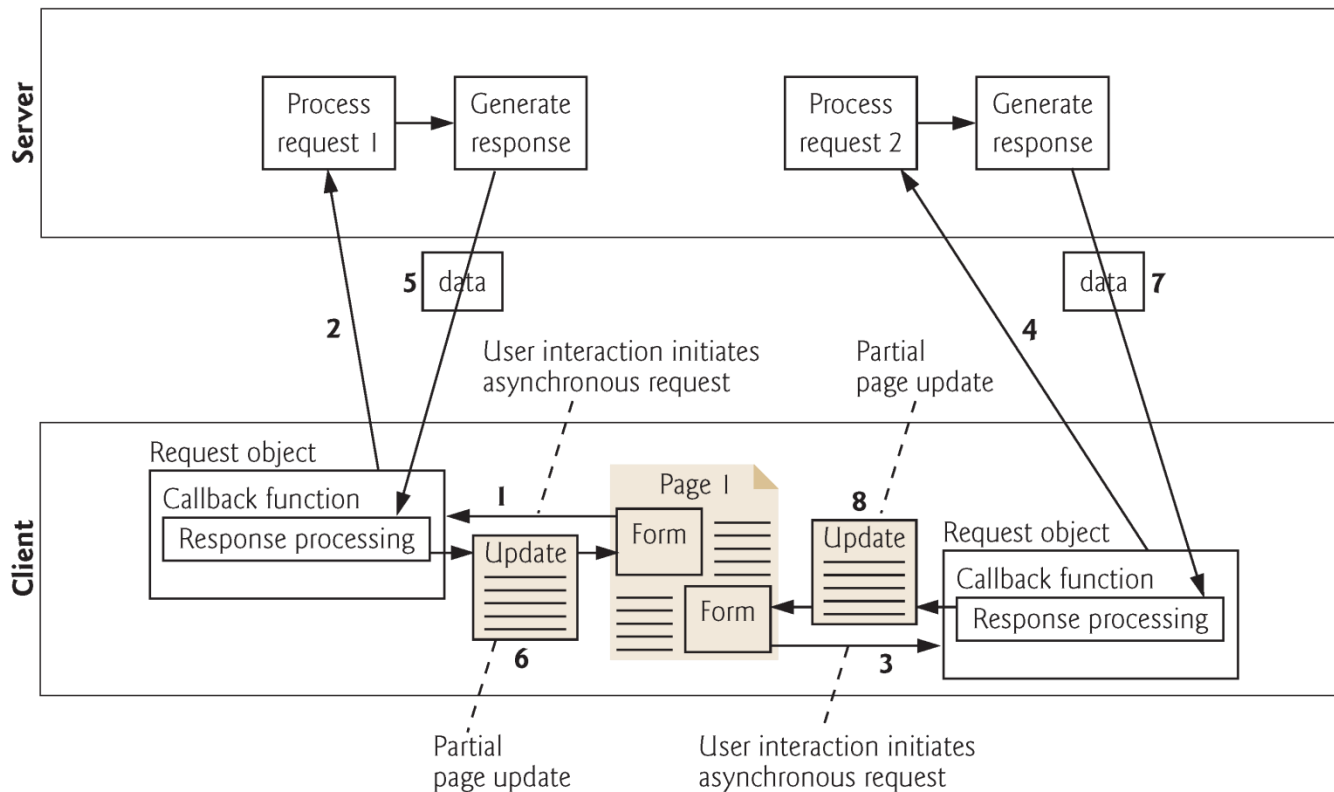
---

**Fig. 30.8** | Form for adding an entry to an address book. (Part 3 of 3.)



**Fig. 30.9** | Classic web application reloading the page for every user interaction.





**Fig. 30.10** | Ajax-enabled web application interacting with the server asynchronously.

a) Submitting the form before entering any information



Validating Form Data

localhost:8080/ValidationAjax/

Java SE 6 API Java SE 7 API Other bookmarks

**Please fill out the following form:**

All fields are required and must contain valid information

Name:

E-mail:

Phone:

**Fig. 30.11** | JSP that demonstrates validation of user input. (Part 1 of 4.)

b) Error messages displayed after submitting the empty form

The screenshot shows a web browser window titled "Validating Form Data". The address bar displays "localhost:8080/ValidationAjax/". Below the address bar, there are two tabs labeled "Java SE 6 API" and "Java SE 7 API", and a bookmark labeled "Other bookmarks". The main content area of the browser displays the following text:

**Please fill out the following form:**

All fields are required and must contain valid information

Name:  Please enter your name

E-mail:  Please enter a valid e-mail address

Phone:  Please enter a valid phone number

Submit

**Fig. 30.11** | JSP that demonstrates validation of user input. (Part 2 of 4.)

c) Error messages displayed after submitting invalid information

The screenshot shows a web browser window titled "Validating Form Data" with the address bar displaying "localhost:8080/ValidationAjax/". The browser's bookmark bar includes "Java SE 6 API", "Java SE 7 API", and "Other bookmarks". The main content area displays the heading "Please fill out the following form:" followed by the instruction "All fields are required and must contain valid information". Below this, there are three input fields with associated error messages in red text:

- Name:** The input field contains "Paul plus a bunch of other". To its right, the error message "Name must be fewer than 30 characters" is displayed.
- E-mail:** The input field contains "not a valid email". To its right, the error message "Invalid e-mail address format" is displayed.
- Phone:** The input field contains "55-1234". To its right, the error message "Invalid phone number format" is displayed.

At the bottom left of the form area, there is a "Submit" button with a mouse cursor hovering over it.

**Fig. 30.11** | JSP that demonstrates validation of user input. (Part 3 of 4.)

c) Error messages displayed after submitting invalid information

Validating Form Data

localhost:8080/ValidationAjax/

Java SE 6 API Java SE 7 API Other bookmarks

## Please fill out the following form:

All fields are required and must contain valid information

Name:  Name must be fewer than 30 characters

E-mail:  Invalid e-mail address format

Phone:  Invalid phone number format

**Fig. 30.11** | JSP that demonstrates validation of user input. (Part 4 of 4.)



```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- index.xhtml -->
4  <!-- Validating user input -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8      xmlns:h="http://java.sun.com/jsf/html"
9      xmlns:f="http://java.sun.com/jsf/core">
10     <h:head>
11         <title>Validating Form Data</title>
12         <h:outputStylesheet name="style.css" library="css"/>
13     </h:head>
14     <h:body>
15         <h:form>
16             <h1>Please fill out the following form:</h1>
17             <p>All fields are required and must contain valid information</p>
18             <h:panelGrid columns="3">
19                 <h:outputText value="Name:"/>
20                 <h:inputText id="nameInputText" required="true"
21                     requiredMessage="Please enter your name"
22                     value="#{validationBean.name}"
23                     validatorMessage="Name must be fewer than 30 characters">
24                     <f:validateLength maximum="30" />
```

**Fig. 30.12** | Ajax enabling the Validation app. (Part I of 3.)



```
25      </h:inputText>
26      <h:message id="nameMessage" for="nameInputText"
27      styleClass="error"/>
28      <h:outputText value="E-mail:"/>
29      <h:inputText id="emailInputText" required="true"
30      requiredMessage="Please enter a valid e-mail address"
31      value="#{validationBean.email}"
32      validatorMessage="Invalid e-mail address format">
33      <f:validateRegex pattern=
34      "\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*" />
35      </h:inputText>
36      <h:message id="emailMessage" for="emailInputText"
37      styleClass="error"/>
38      <h:outputText value="Phone:"/>
39      <h:inputText id="phoneInputText" required="true"
40      requiredMessage="Please enter a valid phone number"
41      value="#{validationBean.phone}"
42      validatorMessage="Invalid phone number format">
43      <f:validateRegex pattern=
44      "((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}" />
45      </h:inputText>
46      <h:message id="phoneMessage" for="phoneInputText"
47      styleClass="error"/>
```

**Fig. 30.12** | Ajax enabling the Validation app. (Part 2 of 3.)



```
48         </h:panelGrid>
49         <h:commandButton value="Submit">
50             <f:ajax execute="nameInputText emailInputText phoneInputText"
51                 render=
52                 "nameMessage emailMessage phoneMessage resultOutputText"/>
53         </h:commandButton>
54         <h:outputText id="resultOutputText" escape="false"
55             value="#{validationBean.response}"/>
56     </h:form>
57 </h:body>
58 </html>
```

**Fig. 30.12** | Ajax enabling the Validation app. (Part 3 of 3.)