

# Handling User Input

---



**Jesper de Jong**  
SOFTWARE ARCHITECT

@jesperdj

[www.jesperdj.com](http://www.jesperdj.com)



# Overview



## User Input

- Forms and User Input Components

## Validation

- Standard and Custom Validators
- Bean Validation

## Conversion

- Standard and Custom Converters

## The Request Processing Lifecycle

## Events and Listeners

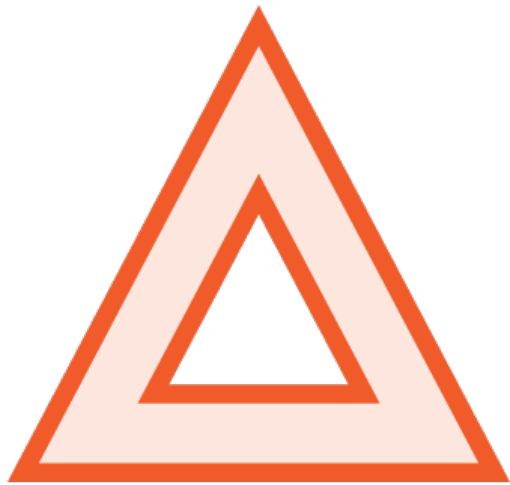


# Demo



## Creating the Sign-in Page

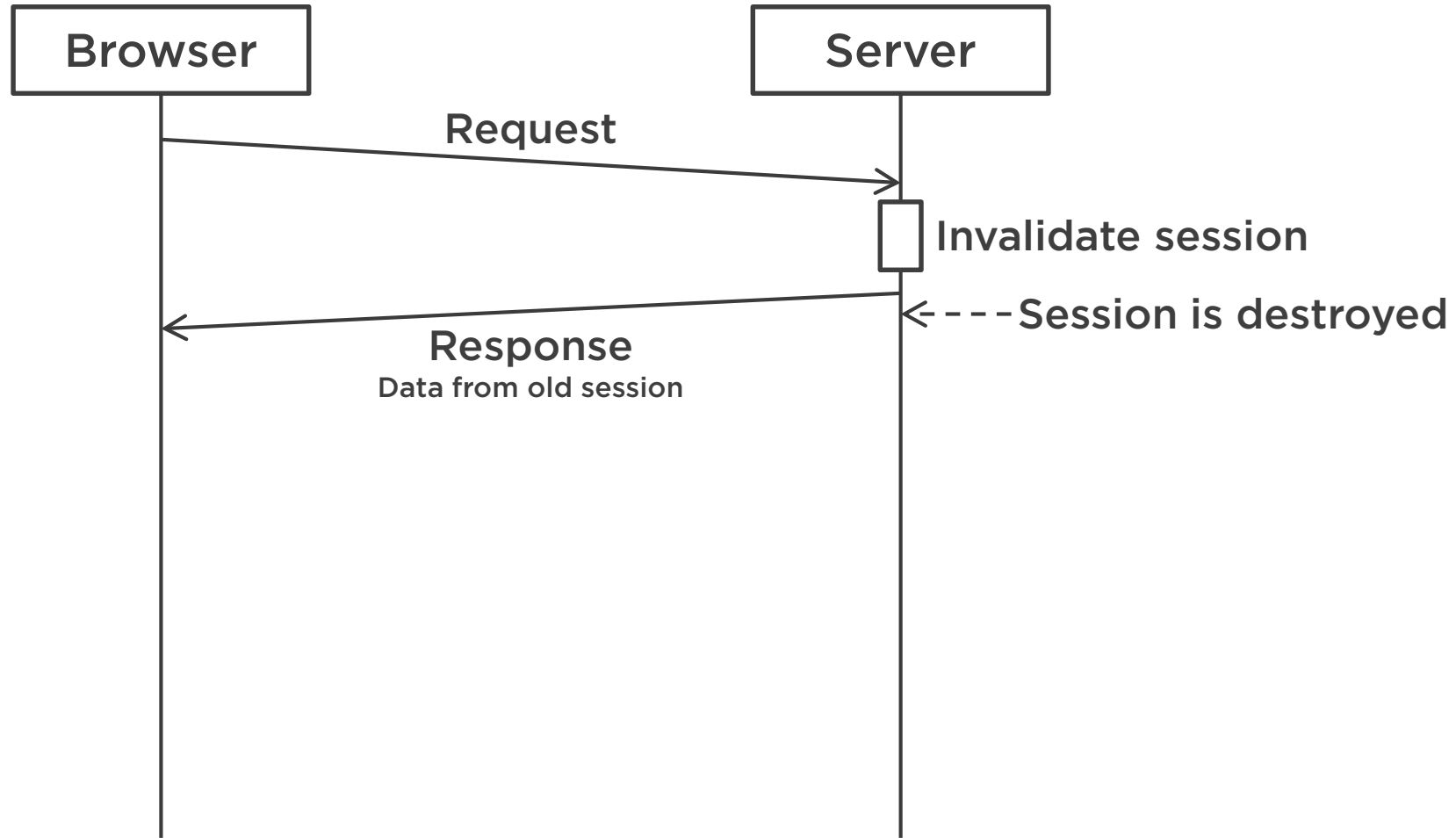




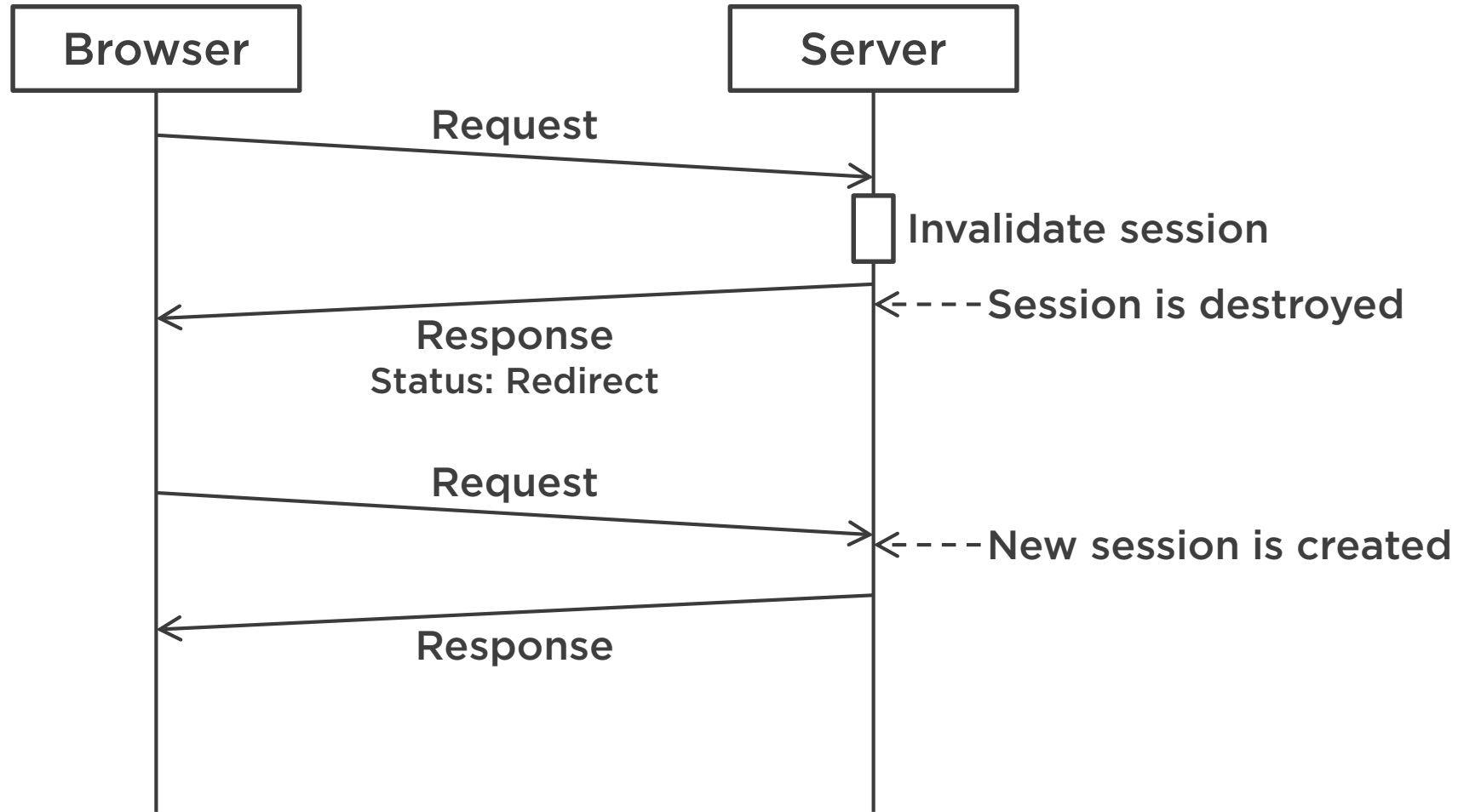
This is not a good example of  
web application security



# Sign-out Sequence



# Sign-out Sequence



# Demo



## Creating the User Details Page



# Validation

## Using the Standard Validators





# Standard Validators

- **f:validateRequired**
  - Check that a required field is not left empty
- **f:validateLength**
  - Check for minimum or maximum number of characters
- **f:validateRegex**
  - Check that a field value matches a regular expression



# Standard Validators

- **f:validateLongRange**
  - Check that a field contains an integer in a range
- **f:validateDoubleRange**
  - Check that a field contains a floating-point number in a range
- **f:validateBean**
  - Specify options for Bean Validation



# Validating a Required Field

```
<h:inputText id="username" value="#{signIn.username}">  
    <f:validateRequired/>  
</h:inputText>
```

```
<h:inputText id="username" value="#{signIn.username}"  
    required="true" />
```



# Validating a Required Field

```
<h:inputText id="username" value="#{signIn.username}">  
    <f:validateRequired/>  
</h:inputText>
```

```
<h:inputText id="username" value="#{signIn.username}"  
    required="#{booleanExpression}" />
```

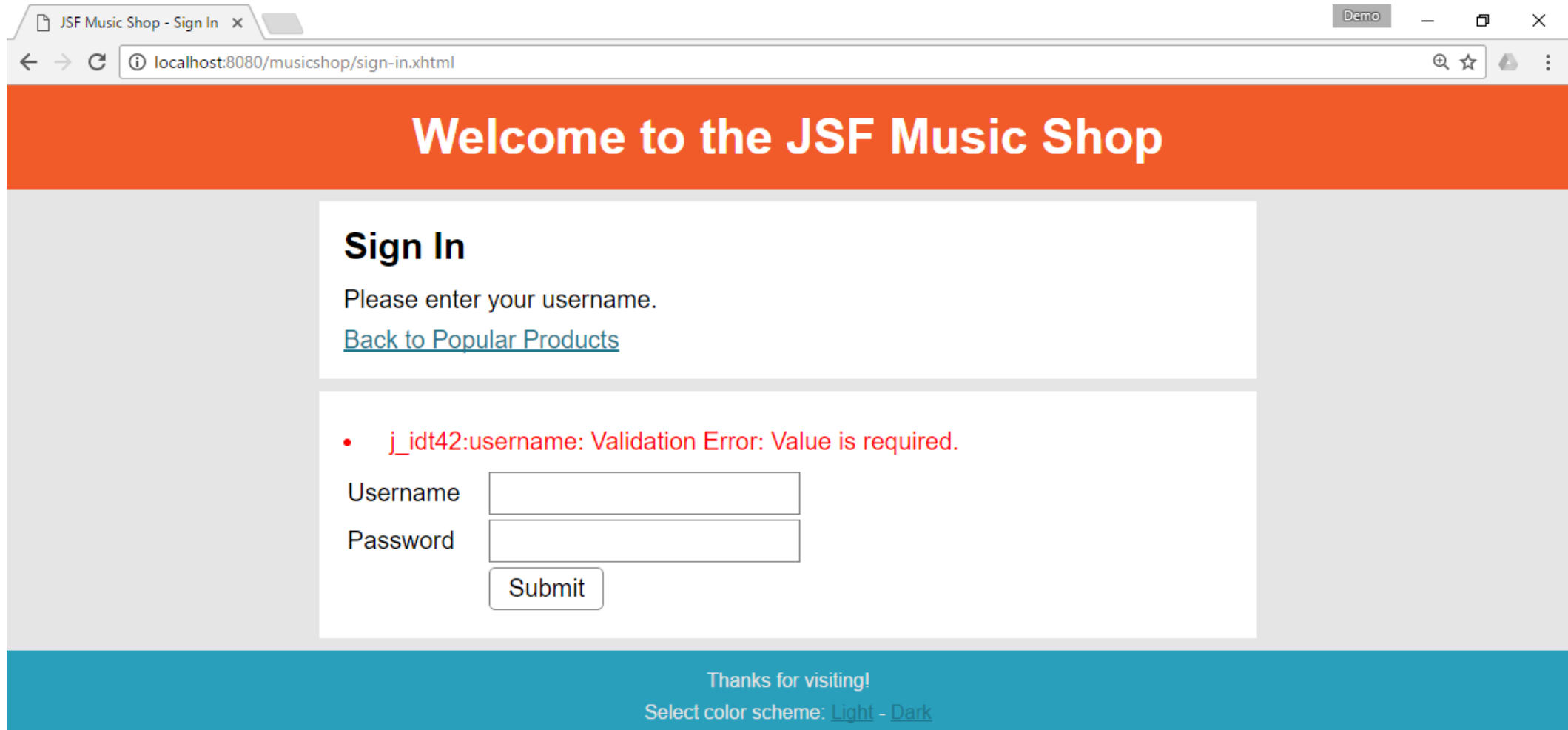


# Displaying Validation Errors

- **h:messages**
  - Show all validation errors in a list or table
- **h:message**
  - Show a validation error for a specific component



# Displaying Validation Errors



The screenshot shows a web browser window with the title "JSF Music Shop - Sign In". The address bar displays "localhost:8080/musicshop/sign-in.xhtml". The page features an orange header with the text "Welcome to the JSF Music Shop". Below this is a white sign-in form with the heading "Sign In" and the instruction "Please enter your username." A blue link "Back to Popular Products" is present. A red validation error message is displayed: "j\_idt42:username: Validation Error: Value is required." The form includes input fields for "Username" and "Password", and a "Submit" button. The footer is a blue bar with the text "Thanks for visiting!" and a link to "Select color scheme: Light - Dark".

JSF Music Shop - Sign In

localhost:8080/musicshop/sign-in.xhtml

## Welcome to the JSF Music Shop

### Sign In

Please enter your username.

[Back to Popular Products](#)

- j\_idt42:username: Validation Error: Value is required.

Username

Password

Thanks for visiting!

Select color scheme: [Light](#) - [Dark](#)

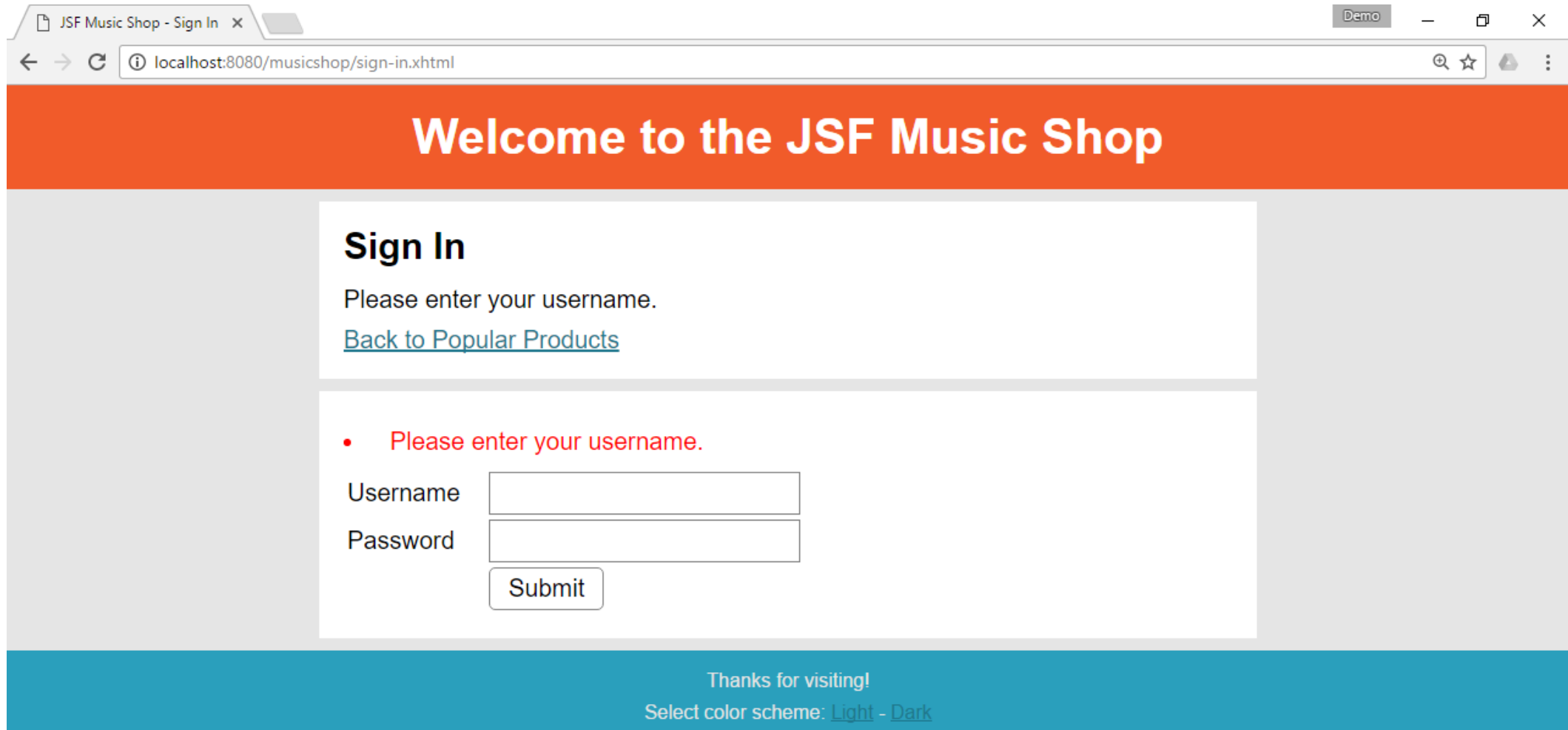


# Specifying Validator Messages

```
<h:inputText id="username" value="#{signIn.username}"  
    required="true"  
    requiredMessage="Please enter your username." />
```



# Displaying Validation Errors



The screenshot shows a web browser window with the title "JSF Music Shop - Sign In". The address bar displays "localhost:8080/musicshop/sign-in.xhtml". The page features an orange header with the text "Welcome to the JSF Music Shop". Below this, there is a "Sign In" section with the instruction "Please enter your username." and a link "Back to Popular Products". A red validation error message, "Please enter your username.", is displayed above the username input field. The form includes fields for "Username" and "Password", and a "Submit" button. The footer contains the text "Thanks for visiting!" and a link to "Select color scheme: Light - Dark".

JSF Music Shop - Sign In

localhost:8080/musicshop/sign-in.xhtml

**Welcome to the JSF Music Shop**

**Sign In**

Please enter your username.

[Back to Popular Products](#)

- Please enter your username.

Username

Password

Thanks for visiting!

Select color scheme: [Light](#) - [Dark](#)





# Validating Using a Regular Expression

```
<h:inputText id="username" value="#{signIn.username}"  
             validatorMessage="Please enter ...">  
    <f:validateRegex pattern="[A-Za-z0-9]{2,20}" />  
</h:inputText>
```



# Adding Validation Errors Programmatically

```
if (/* invalid username or password */) {  
    FacesContext.getCurrentInstance().addMessage(null,  
        new FacesMessage("Please enter a valid username and password."));  
    return "sign-in";  
}
```



# Demo



## Implementing Custom Validators



# US Phone Number Rules

- **Format: AAA-OOO-NNNN**
- **AAA = Area code, 3 digits**
  - First digit must not be 0 or 1
- **OOO = Office code, 3 digits**
  - First digit must not be 0 or 1
  - Second and third digits must not both be 1
- **NNNN = Subscriber number, 4 digits**



Validation

**Using Bean Validation**



# Bean Validation

- Specify constraints on fields of Java beans
- **@NotNull**
  - Check that a field is not null
- **@Size**
  - Check that the number of characters is in a range
- **@Pattern**
  - Check that a field matches a regular expression
- **@Min** and **@Max**
  - Check that an integer value is in a range
- **@Past** and **@Future**
  - Check that a date value is in the past or the future



# Bean Validation

- JSF validation: Specify constraints in the **view**
- Bean Validation: Specify constraints in the **model**



# Bean Validation

- **JSF validation: Specify constraints in the **view****
  - The view is the appropriate place to specify error messages
- **Bean Validation: Specify constraints in the **model****
  - Close to where data is stored
  - Only need to specify constraints in one place



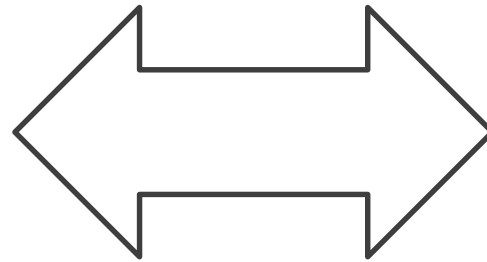


# Conversion Between View and Model

## Facelets User Interface



The diagram illustrates a Facelets User Interface. It consists of a central white rectangular area containing the text "User Details". Below this title are two input fields: "Name" and "Birth Date". The entire form is enclosed within a larger container that has an orange header bar at the top and a blue footer bar at the bottom. The form itself is framed by a light gray border.



## Managed Beans Model

```
@Named
@ViewScoped
public class UserDetails {
    private String name;
    private Date birthDate;
    // ...
}
```

# Example: Product Price

```
public class Product {  
    private BigDecimal price;  
    // ...  
}
```

Price: \$ `{productDetails.product.price}`

Price:

```
<h:outputText value="{productDetails.product.price}">  
    <f:convertNumber type="currency" currencySymbol="$" />  
</h:outputText>
```



## Example: Birth Date

```
<h:inputText id="birthDate" p:type="date"  
             value="#{userDetails.user.birthDate}">  
    <f:convertDateTime pattern="yyyy-MM-dd" />  
</h:inputText>
```



# Standard Converters

Byte  
Short  
Long  
Integer

Float  
Double  
Number

BigDecimal  
BigInteger

Boolean  
Character  
Date  
Enum



# Demo



## Implementing a Phone Number Converter



# The Request Processing Lifecycle

---



# Lifecycle Phases

1. Restore View
2. Apply Request Values
3. Process Validations
4. Update Model Values
5. Invoke Application

**Execute**

6. Render Response

**Render**



# The FacesContext Object

- The **FacesContext** object stores all context information for processing the current request





# Two Kinds of Requests

- **Initial request**
  - When a user first enters a page
- **Postback request**
  - When a form is posted to the URL of the current page
  - For example: User clicks a link or button from an `h:commandLink` or `h:commandButton`



# Initial Request Lifecycle

## 1. Restore View

- Create a new, empty component tree

2. Apply Request Values

3. Process Validations

4. Update Model Values

5. Invoke Application

} Not executed for an  
initial request

## 6. Render Response

- Populate the component tree
- Render the page (generate HTML)
- Save the component tree



# Postback Request Lifecycle

## 1. Restore View

- Restore the component tree

## 2. Apply Request Values

- Get form fields, headers, cookies etc. from the request

## 3. Process Validations

- Call converters and validators

## 4. Update Model Values

- Update managed bean fields with component values

## 5. Invoke Application

- Execute action methods

## 6. Render Response

- Populate the component tree, generate HTML, save the component tree



# Postback Request Lifecycle

## 1. Restore View

- Restore the component tree

## 2. Apply Request Values

- Get form fields, headers, cookies etc. from the request

## 3. Process Validations

- Call converters and validators

## 4. Update Model Values

- Update managed bean fields with component values

## 5. Invoke Application

- Execute action methods

## 6. Render Response

- Populate the component tree, generate HTML, save the component tree



# Postback Request Lifecycle

- Navigation happens on return value of action method
  - Component tree replaced for new view
- Lifecycle may be cut short by calling special methods on the **FacesContext**
  - **renderResponse()** - Jump to Render Response
  - **responseComplete()** - Jump to the end



# Implementing Event Listeners

---



# Implementing Event Listeners

- **Great way to learn the details of the lifecycle**
- **Application events**
- **System events**



# Application Events

- **Generated by UI components**
- **Action events**
  - For example: Button clicked
- **Value change events**
  - For example: Text field value modified
- **Custom application events**





# Adding a Listener for Action Events

```
<h:commandButton id="submit" value="Submit"  
    actionListener="#{myBean.actionPerformed}" />
```

```
@Named
```

```
@RequestScoped
```

```
public class MyBean {
```

```
    public void actionPerformed(ActionEvent event) {
```

```
        // ...
```

```
    }
```

```
}
```



# Adding a Listener for Action Events

```
<h:commandButton id="submit" value="Submit">  
    <f:actionListener type="com.mypackage.MyListener" />  
</h:commandButton>
```

```
public class MyListener implements ActionListener {  
    public void processAction(ActionEvent event) {  
        // ...  
    }  
}
```



# Adding a Listener for Value Change Events

```
<h:inputText id="username" value="#{signIn.username}"  
    valueChangeListener="#{myBean.valueChanged}">
```

```
<h:inputText id="username" value="#{signIn.username}">  
    <f:valueChangeListener type="com.mypackage.MyListener" />  
</h:inputText>
```



# System Events

- System-level events
- Register listeners using `<f:event>`
  - Attribute `type` is the name of the event type
  - Attribute `listener` refers to a method to handle the event
- System event types
  - `postAddToView`
  - `preValidate`
  - `postValidate`
  - `preRenderView`
  - `preRenderComponent`



# Phase Listener

- Called at the start and end of some or all of the lifecycle phases
- Must be declared in Faces configuration file



# Summary



## Forms and User Input Components

### Validation

- Standard and custom validators
- Bean Validation

### Conversion

- Standard and custom converters

### Request Processing Lifecycle

- Initial and postback requests

### Events and Listeners

- Application and system events



# Coming Up



## Navigation

