	UNIVERSIDADDONBOSCO **FACULTADDEESTUDIOSTECNOLÓGICOS** **COORDINACIÓNDECOMPUTACIÓN**
Ciclo II	**Desarrollo de aplicaciones con Web Frameworks** **Guía de Laboratorio No. 12** **JSF, JDBC y uso de AJAX**

I.Objetivos

Que el alumno utilice JDBC con el Framework JSF.

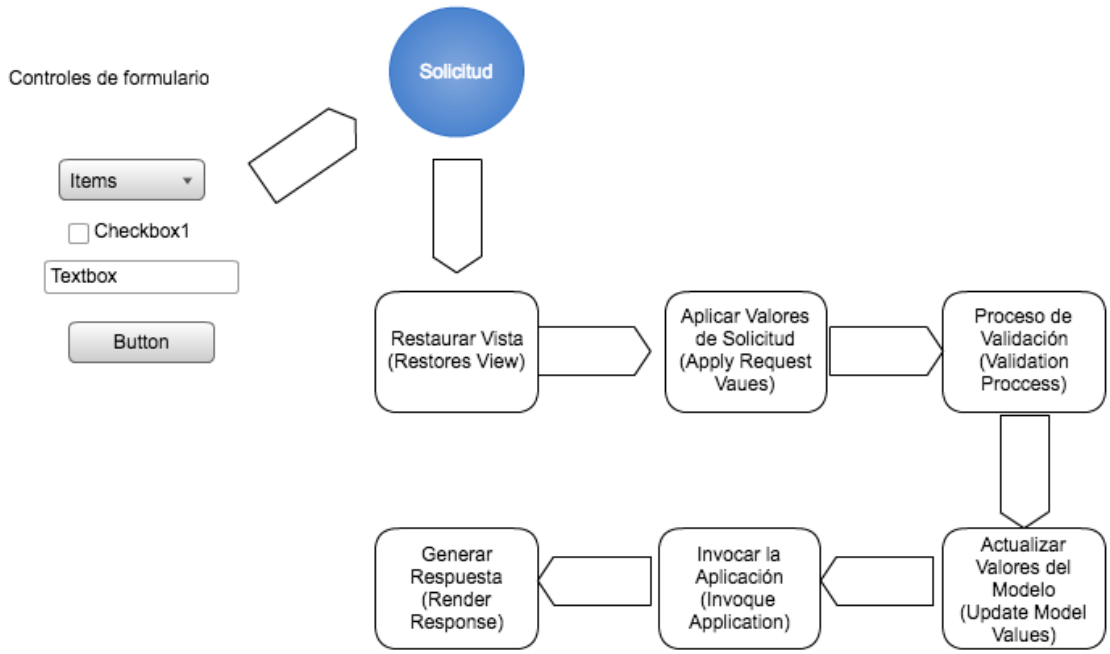
Que el alumno identifique las "3 capas" del model MVC.

Que el alumno sea capaz de crear aplicaciones asíncronas a partir de las tags de jsf.

Que el alumno comprenda El Ciclo de Vida de una aplicación JSF

II. INTRODUCCIÓN

Toda aplicación Web que tiene el framework JSF realiza una secuencia específica que se le denomina Ciclo de Vida de JSF. Consiste en una serie de pasos con el cual procesa las peticiones, realiza los procesos y retorna los resultados. A continuación puede visualizarse cómo funciona El Ciclo de Vida:



1. Restaurar la vista (restore view). En este paso se obtiene el árbol de componentes correspondiente a la vista JSF de la petición. Si se ha generado antes se recupera, y si es la primera vez que el usuario visita la página, se genera a partir de la descripción JSF.

2. Aplicar los valores de la petición (apply request values). Una vez obtenido el árbol de componentes, se procesan todos los valores asociados a los mismos. Se convierten todos los datos de la petición a tipos de datos Java y, para aquellos que tienen la propiedad inmediate a cierta, se validan, adelantándose a la siguiente fase.

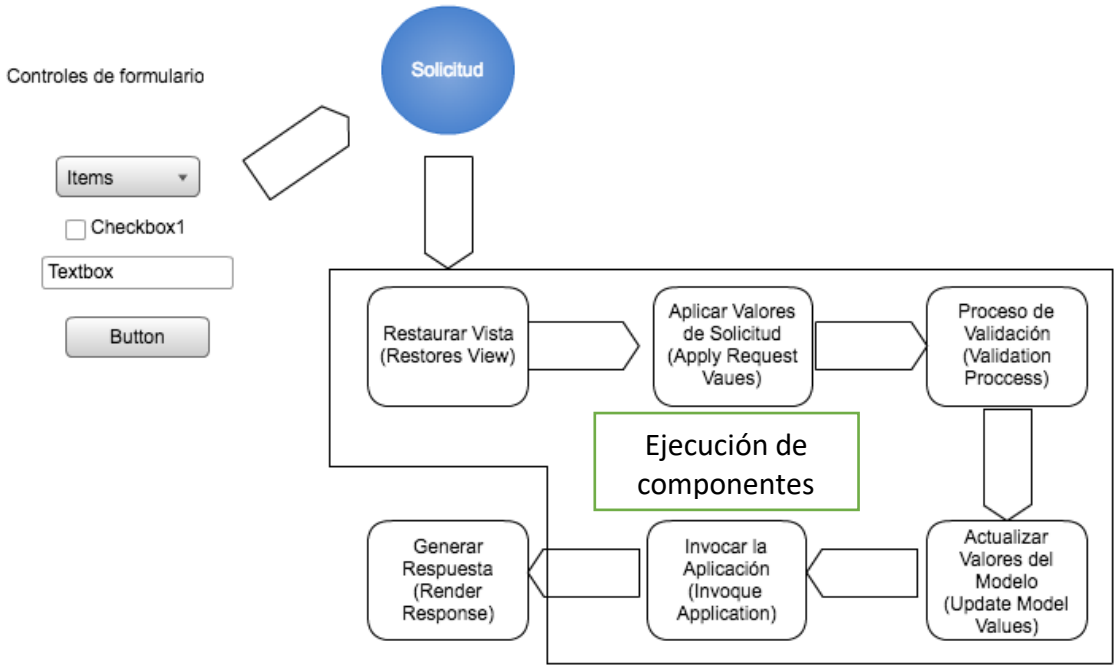
3. Procesar las validaciones (process validations). Se validan todos los datos. Si existe algún error, se encola un mensaje de error y se termina el ciclo de vida, saltando al último paso (renderizar respuesta).

4. Actualizar los valores del modelo (update model values). Cuando se llega a esta fase, todos los valores se han procesado y se han validado. Se actualizan entonces las propiedades de los beans gestionados asociados a los componentes.

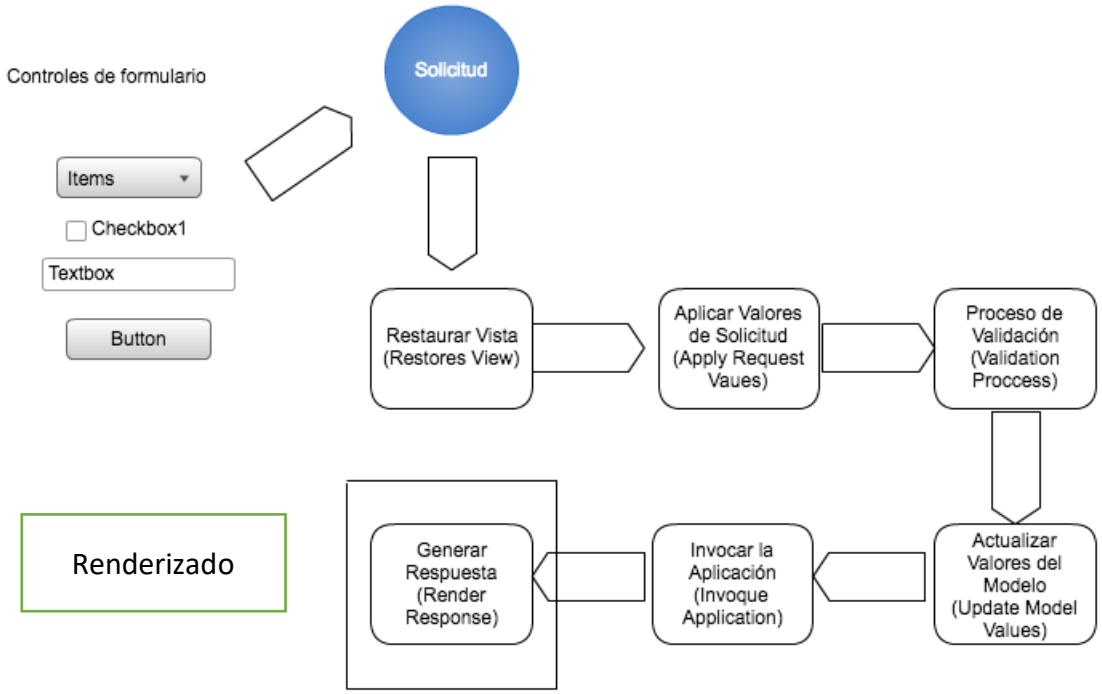
5. Invocar a la aplicación (invoke application) . Cuando se llega a esta fase, todas las propiedades de los beans asociados a componentes de entrada (input) se han actualizado. Se llama en este momento a la acción seleccionada por el usuario.

6. Renderizar la respuesta (render response).

JSF permite incrustar AJAX (asynchronous javascript and xml) a las aplicaciones, ejecutando todo el proceso del ciclo de vida menos uno, el renderizado de toda la página web.



La parte de ejecución hace referencia como su nombre indica a todo el ciclo de vida que está ligado a la propia ejecución de los componentes dejando de lado la última fase, la fase de renderizado. En cambio la segunda fase es opuesta a la primera solo se encarga del renderizado.

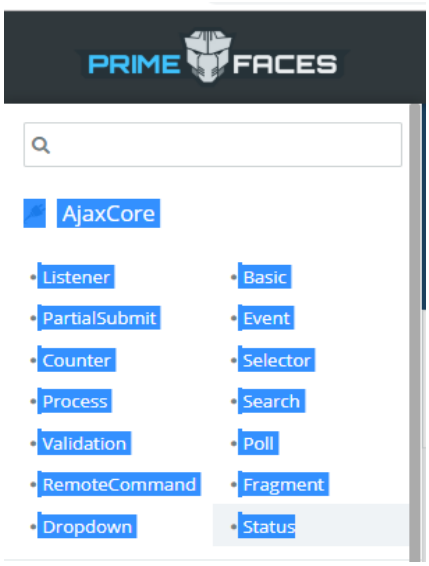


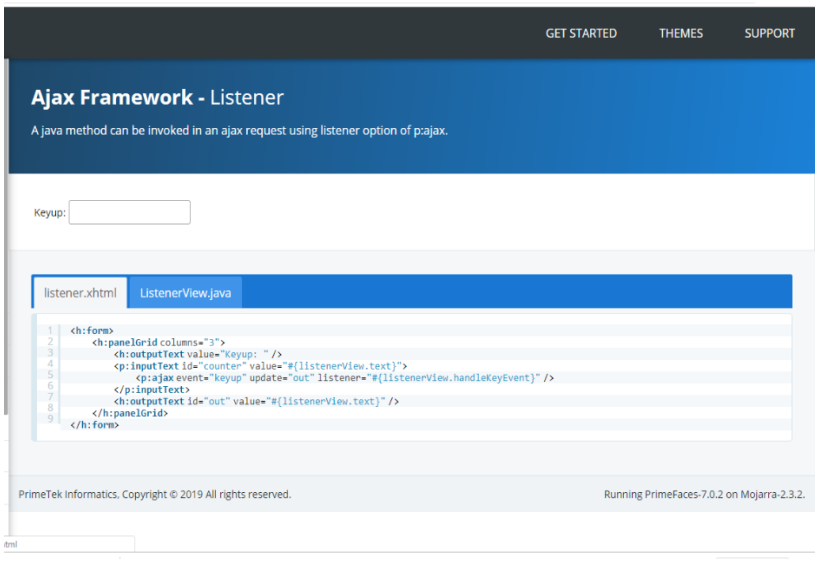
JSF y AJAX permite aplicar estas fases a controles diferentes y obtener un resultado. Por ejemplo puede hacer que el control de h:inputText pase por todas las fases de ejecución, pero que no le haga falta pasar por la fase de render ya que esta renderizado correctamente. También puede hacer que un control h:outputText funcione de la forma contraria y no pase por ninguna fase salvo la última.

III. Desarrollo

Antes de iniciar con la práctica por favor visitar el siguiente sitio para realizar prácticas y comprender el funcionamiento de AJAX.

https://www.primefaces.org/showcase/ui/ajax/listener.xhtml





Tag Attributes:

S.No	Attribute & Description
1	**disabled** Si es verdadero, el comportamiento Ajax se aplicará a cualquier componente principal o secundario. Si es falso, se deshabilitará el comportamiento de Ajax.
2	**Event** El evento que invocará las solicitudes de Ajax, por ejemplo, "clic", "cambio", "desenfoque", "pulsación de tecla", etc.
3	**Execute** Una lista separada por espacios de ID para componentes que deben incluirse en la solicitud Ajax.
4	**Immediate** Si los eventos de comportamiento "verdadero" generados a partir de este comportamiento se transmiten durante la fase de Aplicar valores de solicitud. De lo contrario, los eventos se transmitirán durante la fase de Invocar Aplicaciones.
5	**Listener** Una expresión EL para un método en un bean de respaldo que se llamará durante la solicitud Ajax.
6	**Onerror** El nombre de una función de devolución de llamada de JavaScript que se invocará si hay un error durante la solicitud de Ajax.

7	**Onevent** El nombre de una función de devolución de llamada de JavaScript que se invocará para manejar eventos de IU.
8	**Render** Una lista separada por espacios de identificadores para componentes que se actualizarán después de una solicitud Ajax.

Antes de iniciar con la práctica se deben de conocer los siguientes términos:

Using Annotation

```
importar  javax.faces.bean.ManagedBean;
importar  javax.faces.bean.RequestScoped;

@ManagedBean // Usando la anotación ManagedBean
@RequestScoped // Usando la anotación del alcance
 Usuario de clase pública {
     nombre de la cadena privada ;
     Cadena pública getName () {
         nombre de   retorno
    }
    nombre de conjunto público vacío  (nombre de cadena) {
     este .nombre = nombre;
}
}
```

@ManagedBean Annotation

Uso de anotaciones para configurar beans gestionados

El @ManagedBean (javax.faces.bean.ManagedBean anotación) en una clase que registra automáticamente la clase como un recurso con la implementación JavaServer Faces.

El fragmento de código anterior muestra un bean que está gestionado por la implementación de **JavaServer Faces** y está disponible durante la duración de la sesión.

No es necesario configurar la instancia de bean administrado en el archivo **faces-config.xml** . En efecto, esta es una alternativa al enfoque del archivo de recursos de configuración de la aplicación y reduce la tarea de configurar beans administrados.

También puede definir el alcance del bean administrado dentro del archivo de clase, como se muestra en el ejemplo anterior. Puede anotar beans con solicitud, sesión, aplicación o vista de alcance.

Eager Managed Bean

El bean administrado es perezoso por defecto. Esto significa que el bean se crea una instancia solo cuando se realiza una solicitud desde la aplicación.

Puede forzar la creación de una instancia de un bean y colocarlo en el ámbito de la aplicación tan pronto como se inicie la aplicación. Debe establecer el atributo eager del bean administrado en verdadero como se muestra en el siguiente ejemplo:

```
@ManagedBean(eager=true)
```

Using Managed Bean Scopes

Puede usar anotaciones para definir el alcance en el que se almacenará el bean. Puede especificar uno de los siguientes ámbitos para una clase de bean:

Aplicación (@ApplicationScoped): el alcance de la aplicación persiste en todas las interacciones de los usuarios con una aplicación web.

Sesión (@SessionScoped): el alcance de la sesión persiste en varias solicitudes HTTP en una aplicación web.

Vista (@ViewScoped): el alcance de la vista persiste durante la interacción de un usuario con una sola página (vista) de una aplicación web.

Solicitud (@RequestScoped): el alcance de la solicitud persiste durante una única solicitud HTTP en una aplicación web.

Ninguno (@NoneScoped): indica que un ámbito no está definido para la aplicación.

Personalizado (@CustomScoped): un alcance no estándar definido por el usuario. Su valor debe configurarse como java.util.Map . Los ámbitos personalizados se utilizan con poca frecuencia.

Es posible que desee utilizar @**NoneScoped** cuando un bean administrado haga referencia a otro bean administrado. El segundo bean no debe estar en un ámbito (@**NoneScoped**) si se supone que se debe crear solo cuando se hace referencia a él. Si define un bean como @**NoneScoped** , el bean se crea una instancia de nuevo cada vez que se hace referencia, por lo que no se guarda en ningún ámbito.

Si el atributo de enlace de una etiqueta de componente hace referencia a su bean administrado , debe definir el bean con un alcance de solicitud. Si colocara el bean en la sesión o en el ámbito de la aplicación, el bean tendría que tomar precauciones para garantizar la seguridad del hilo, ya que las instancias de **javax.faces.component.UIComponent** dependen de la ejecución dentro de un solo hilo.

Si está configurando un bean que permite asociar atributos con la vista, puede usar el alcance de la vista. Los atributos persisten hasta que el usuario haya navegado a la siguiente vista.

Ejemplo práctico:

JSF proporciona un excelente soporte para realizar llamadas ajax. Proporciona la etiqueta **f: ajax para manejar las llamadas ajax.**

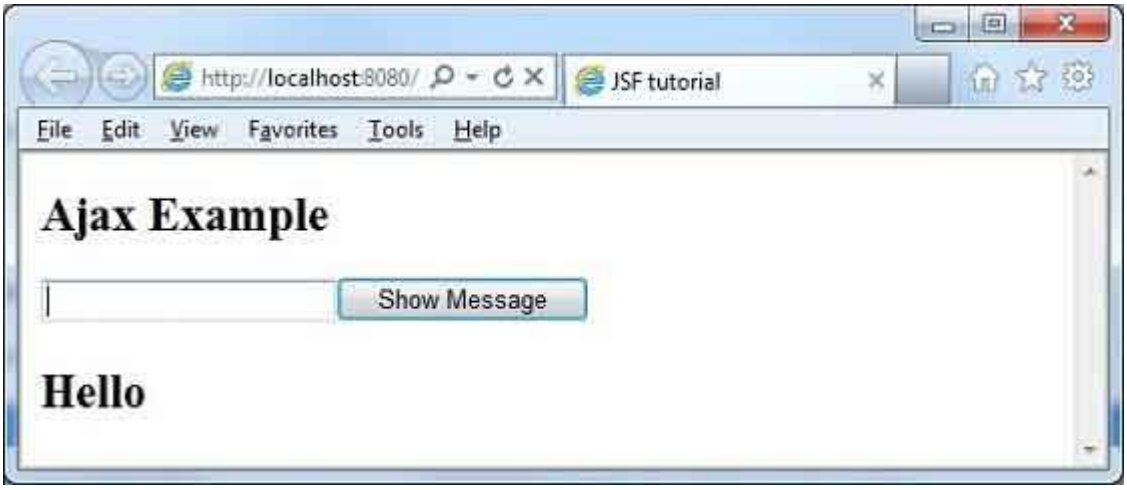
Codificar el siguiente beans: **package / com.udb.test / UserData.java**

```
import java.io.Serializable;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {
   private static final long serialVersionUID = 1L;
   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getWelcomeMessage() {
      return "Hello " + name;
   }
}
```

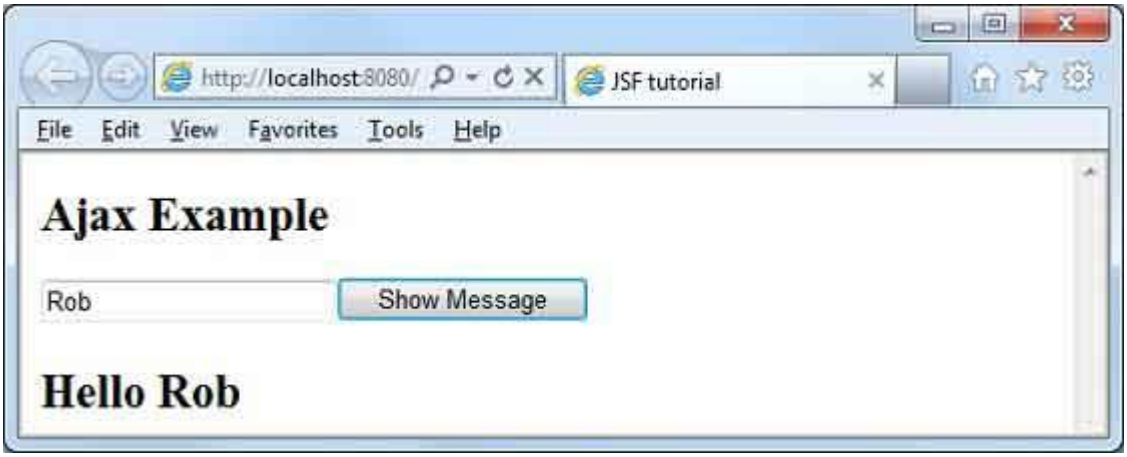
Crear la siguiente página: **home.xhtml**

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns = "http://www.w3.org/1999/xhtml"
   xmlns:h = "http://java.sun.com/jsf/html"
   xmlns:f = "http://java.sun.com/jsf/core"
   xmlns:tp = "http://java.sun.com/jsf/composite/tutorialspoint">

   <h:head>
      <title>JSF tutorial</title>
   </h:head>

   <h:body>
      <h2>Ajax Example</h2>

      <h:form>
         <h:inputText id = "inputName" value =
"#{userData.name}"></h:inputText>
         <h:commandButton value = "Show Message">
            <f:ajax execute = "inputName" render = "outputMessage" />
         </h:commandButton>
         <h2><h:outputText id = "outputMessage"
            value = "#{userData.welcomeMessage != null ?
            userData.welcomeMessage : ''}"
         /></h2>
      </h:form>
   </h:body>
</html>
```

Si todo está bien con su aplicación, esto producirá el siguiente resultado.



Ingrese el nombre y presione el botón Mostrar mensaje. Verá el siguiente resultado sin la actualización de la página / envío del formulario.



Cree una base de datos denominada Authors en mysql a partir del siguiente script:

```
create database authors;

use authors;

create table literarygenre(
id int primary key auto_increment,
name varchar(20));

create table authors(
id int primary key auto_increment,
name varchar(60),
birth date,
number varchar(30),
literarygenreid int,
constraint ck_authors
foreign key (literarygenreid) references literarygenre(id)
);

insert into literarygenre (name) values
('Drama'),('Poem'),('Prose'),('Horror');

insert into authors (name, birth, number, literarygenreid) values
('Edgar Allan Poe',STR_TO_DATE('19-01-1980','%d-%m-%Y'),'2222-2222',4),
('William Shakespeare',STR_TO_DATE('01-01-1564','%d-%m-%Y'),'2344-
5678',1);
```

Crear DataSource en apache tomcat con nombre jdbc/mysql o crearlo en GlassFish
Web Pages / META-INF / context

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/JSFCrud">
<Resource name="jdbc/mysql" auth="Container" type="javax.sql.DataSource"
               maxActive="100" maxIdle="30" maxWait="10000"
               username="root" password=""
driverClassName="com.mysql.jdbc.Driver"
               url="jdbc:mysql://localhost:3306/authors"/>
</Context>
```

Cree dentro del paquete model el POJO Author:

```
package sv.edu.udb.jsfcrud.model;

import java.util.Date;


public class Author {
    private int authorId;
    private String authorName;
    private Date authorBirth;
    private String authorNumber;
private String literaryGenre;
```

```
    //generar constructor con parámetros y sin parámetros con ayuda del
ide


public Author() {
    }

    public Author(int authorId, String authorName, Date authorAge,
            String authorNumber, String literaryGenre) {
        this.authorId = authorId;
        this.authorName = authorName;
        this.authorBirth = authorAge;
        this.authorNumber = authorNumber;
this.literaryGenre = literaryGenre;
}

    //generar setters y getters con ayuda del ide.

/**
     * @return the authorId
     */
    public int getAuthorId() {
        return authorId;
    }

    /**
     * @param authorId the authorId to set
     */
    public void setAuthorId(int authorId) {
        this.authorId = authorId;
    }

    /**
     * @return the authorName
     */
    public String getAuthorName() {
        return authorName;
    }

    /**
     * @param authorName the authorName to set
     */
    public void setAuthorName(String authorName) {
        this.authorName = authorName;
    }

    /**
     * @return the authorBirth
     */
    public Date getAuthorBirth() {
        return authorBirth;
    }

    /**
     * @param authorBirth the authorBirth to set
```

```
     */
    public void setAuthorBirth(Date authorBirth) {
        this.authorBirth = authorBirth;
    }

    /**
     * @return the authorNumber
     */
    public String getAuthorNumber() {
        return authorNumber;
    }

    /**
     * @param authorNumber the authorNumber to set
     */
    public void setAuthorNumber(String authorNumber) {
this.authorNumber = authorNumber;
    }

    /**
     * @return the literaryGenre
     */
    public String getLiteraryGenre() {
        return literaryGenre;
    }

    /**
     * @param literaryGenre the literaryGenre to set
     */
    public void setLiteraryGenre(String literaryGenre) {
this.literaryGenre = literaryGenre;
    }

}
```

Ahora debe crear la clase Connection.java para la conexión con la base de datos.

```
package sv.edu.udb.jsfcrud.model;

import java.sql.SQLException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;


public class Connection {

    protected DataSource dataSource;
    protected java.sql.Connection con;

    public Connection() {
                try {
            Context ctx = new InitialContext();
```

```
            dataSource =
(DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
        } catch (NamingException e) {
                e.printStackTrace();
        }
}

    protected void connect() throws SQLException{
         //if resource injection is not support, you still can get it
manually.

        if(dataSource == null)
            throw new SQLException("Can't get data source");

            con = dataSource.getConnection();

        if(con == null)
            throw new SQLException("Can't get database connection");
    }

    protected void close() throws SQLException{
con.close();
    }

}
```

Cree la clase AuthorModel.java que tendrá las operaciones con la tabla Author.

```
package sv.edu.udb.jsfcrud.model;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;


public class AuthorModel extends sv.edu.udb.jsfcrud.model.Connection{



    public AuthorModel(){
        super();
    }


    public List<Author> getAuthors() throws SQLException{

       this.connect();

        PreparedStatement ps
                = con.prepareStatement("select a.id as idAuthor, a.name,
a.birth, a.number, l.name as literarygenre "
                        + "from authors a "
                        + "join literarygenre l "
```

```
                        + "on l.id = a.literarygenreid ");

        ResultSet result = ps.executeQuery();

        List<Author> list = new ArrayList<Author>();

        while(result.next()){
            Author author = new Author();
            author.setAuthorId(result.getInt("idAuthor"));
            author.setAuthorName(result.getString("name"));
            author.setAuthorBirth(result.getDate("birth"));
            author.setAuthorNumber(result.getString("number"));
            author.setLiteraryGenre(result.getString("literarygenre"));
            list.add(author);
        }

        this.close();
        return list;

    }

    public int findSameNameAuhor(String name) throws SQLException{

this.connect();

        PreparedStatement ps
                = con.prepareStatement("select count(*) as total "
                        + "from authors a "
                        + " where  name like ? ");

        ps.setString(1, name);
        ResultSet result = ps.executeQuery();

        int count=0;

        while(result.next()){
            count= result.getInt("total");
        }

        this.close();
        return count;
    }

    public void addAuthor(Author author) throws SQLException{

        this.connect();

        if(dataSource == null)
            throw new SQLException("Can't get data source");

        if(con == null)
            throw new SQLException("Can't get database connection");

        PreparedStatement ps
                = con.prepareStatement("insert into authors
(name,birth,number,literarygenreid) "
                        + " values (?,?,?,?)");
```

```
        ps.setString(1, author.getAuthorName());
        ps.setDate(2, new
java.sql.Date(author.getAuthorBirth().getTime()));//convirtiendo fecha
        ps.setString(3, author.getAuthorNumber());
        ps.setString(4, author.getLiteraryGenre());

        ps.execute();

        this.close();

    }

    public void delete(Author author) throws SQLException{
        this.connect();
         if(dataSource == null)
            throw new SQLException("Can't get data source");

        if(con == null)
            throw new SQLException("Can't get database connection");

        PreparedStatement ps
                = con.prepareStatement("delete from authors where id =
?");


ps.setInt(1, author.getAuthorId());

        ps.execute();

this.close();
    }

    //Usted debe implementar esta funcionalidad
public boolean update(Author author){
        throw new java.lang.UnsupportedOperationException("Not supported
yet.");
}

}
```

Cree el POJO LiteraryGenre.java

```
package sv.edu.udb.jsfcrud.model;


public class LiteraryGenre {

    int id;
    String name;

    public LiteraryGenre() {
    }

    public LiteraryGenre(int id, String name) {
        this.id = id;
```

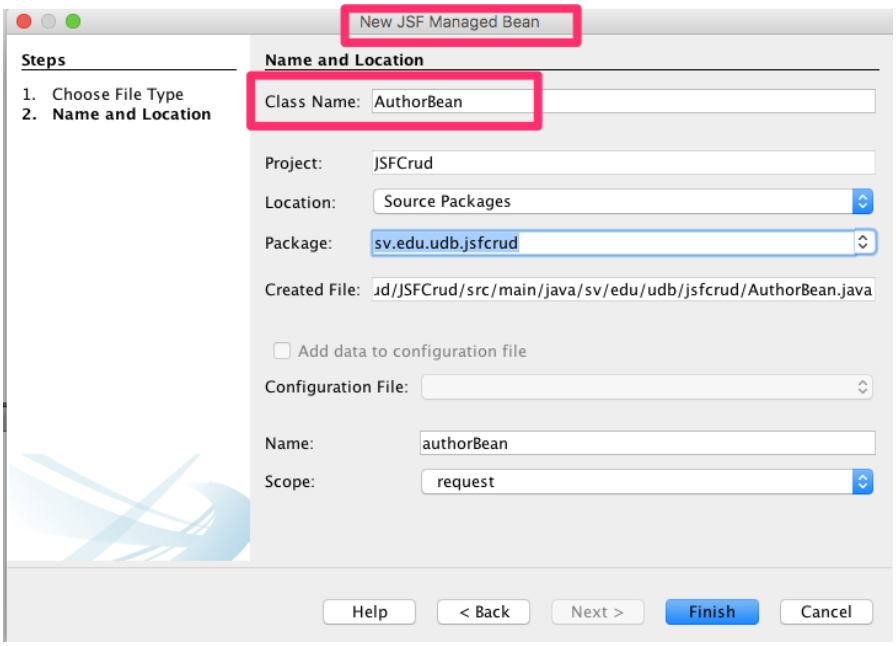
```
        this.name = name;
    }

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

    public int getId() {
        return id;
    }

    public void setName(String name) {
this.name = name;
    }

    public String getName() {
        return name;
}

}
```

Cree el archivo de modelo LiteraryGenreModel.java

```
package sv.edu.udb.jsfcrud.model;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;


public class LiteraryGenreModel extends
sv.edu.udb.jsfcrud.model.Connection{

    public LiteraryGenreModel() {
         super();
    }


     public List<LiteraryGenre> getLiteraryGenre() throws SQLException{

       this.connect();

        PreparedStatement ps
                = con.prepareStatement("select id, name from
literarygenre ");;

        ResultSet result = ps.executeQuery();

        List<LiteraryGenre> list = new ArrayList<LiteraryGenre>();

        while(result.next()){
            LiteraryGenre literaryGenre = new LiteraryGenre();
            literaryGenre.setId(result.getInt("id"));
```

```
            literaryGenre.setName(result.getString("name"));

list.add(literaryGenre);
        }

        this.close();
        return list;

}

}
```

Lo siguiente que debe editar es el ManagedBean y posteriormente el archivo index.xhtml. Comenzaremos con el ManagedBean denominado AuthorBean.java dentro del paquete sv.edu.udb.jsfcrud:




```
package sv.edu.udb.jsfcrud;

import sv.edu.udb.jsfcrud.model.AuthorModel;
import java.io.Serializable;
import java.sql.SQLException;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import sv.edu.udb.jsfcrud.model.Author;
import sv.edu.udb.jsfcrud.model.LiteraryGenreModel;


@ManagedBean
@SessionScoped
public class AuthorBean implements Serializable {
```

```
    private Author author;

    private AuthorModel authorModel = new AuthorModel();

    private LiteraryGenreModel literaryGenreModel = new
LiteraryGenreModel();

    public AuthorBean() {
        this.author = new Author();
    }


    public void addAuthor() throws SQLException{
        authorModel.addAuthor(author);
           FacesContext.getCurrentInstance().addMessage("successMessage",
new FacesMessage(FacesMessage.SEVERITY_INFO, "Agregado Exitosamente",
"Agregado"));
           author = new Author();
    }


    public void deleteAuthor(Author author) throws SQLException{
        authorModel.delete(author);
    }


    public void countAuthor(String name) throws SQLException{
        if(authorModel.findSameNameAuhor(name)>0){
            FacesContext.getCurrentInstance().addMessage("errorMessage",
new FacesMessage(FacesMessage.SEVERITY_INFO, "Este autor ya existe",
"Author"));
        }
    }

    /**
     * @return the author
     */
    public Author getAuthor() {
        return author;
    }

    /**
     * @param author the author to set
     */
    public void setAuthor(Author author) {
        this.author = author;
    }

    /**
     * @return the authorModel
     */
    public AuthorModel getAuthorModel() {
        return authorModel;
    }
```

```
    /**
     * @param authorModel the authorModel to set
     */
    public void setAuthorModel(AuthorModel authorModel) {
        this.authorModel = authorModel;
    }

    /**
     * @return the literaryGenreModel
     */
    public LiteraryGenreModel getLiteraryGenreModel() {
        return literaryGenreModel;
    }

    /**
     * @param literaryGenreModel the literaryGenreModel to set
*/
    public void setLiteraryGenreModel(LiteraryGenreModel
literaryGenreModel) {
        this.literaryGenreModel = literaryGenreModel;
    }



}
```

El formulario tendrá un validador especial SVPhoneValidator.java dentro del paquete sv.edu.udb.util, con él validaremos números de teléfono que son frecuentes en El Salvador.

```
package sv.edu.udb.util;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.FacesValidator;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;


@FacesValidator("sv.edu.udb.util.SVPhoneValidator")
public class SVPhoneValidator implements Validator{

    private static final String CUSTOM_PATTERN = "^[2|3|6|7]{1}\\d{3}-
\\d{4}$";

    private Pattern pattern;
    private Matcher matcher;

    public SVPhoneValidator() {
        pattern = Pattern.compile(CUSTOM_PATTERN);
    }
```

```
    @Override
    public void validate(FacesContext fc, UIComponent uic, Object o)
throws ValidatorException {
    matcher = pattern.matcher(o.toString());
                if(!matcher.matches()){

                        FacesMessage msg =
                                new FacesMessage("Validación de teléfono
falló.",
                                                "Formato incorrecto.");
                        msg.setSeverity(FacesMessage.SEVERITY_ERROR);
                        throw new ValidatorException(msg);

                }
    }

}
```

Ahora edite el archivo index.xhtml. Todas las operaciones Ajax dependen finalmente de esta vista, desde aquí tendrá la implementación que usted considere necesaria para realizar todas las operaciones.

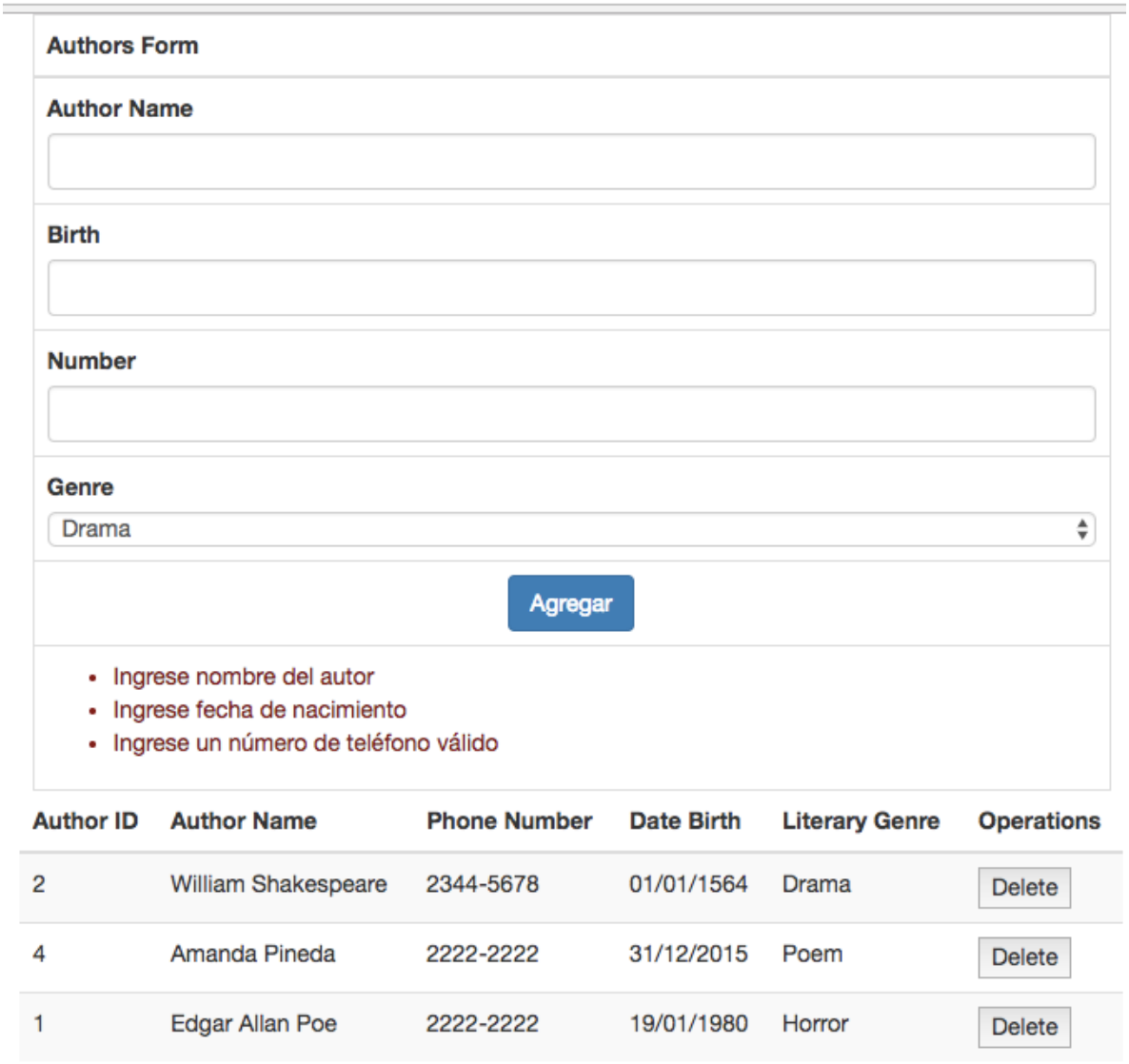
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
<title>Authors</title>
<!-- Este estilo corresponde a boostrap. Puede copiar y pegar del sitio
oficial de bootstrap-->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.c
ss"
      integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
      crossorigin="anonymous" />

<style>
           .table {
                border-radius: 5px;
                width: 50%;
                margin: 0px auto;
                float: none;
            }
</style>
</h:head>
<h:body>
<div class="container-fluid">


<h:form id="authorForm">
<table class="table table-bordered">
```

```
<thead>
<tr>
<th>
                                Authors Form
</th>
</tr>
</thead>
<tbody>
<tr>
<td>

<label for="authorForm:authorName">Author Name</label>
<h:inputText id="authorName" value="#{authorBean.author.authorName}"
required="true"
                                     requiredMessage="Ingrese nombre del
autor" styleClass="form-control">
<f:ajax event="blur"
                                    listener="#{authorBean.countAuthor(au
thorBean.author.authorName)}"
                                    render="@form" />
</h:inputText>


</td>

</tr>
<tr>

<td>
<label for="authorForm:authorBirth">Birth</label>
<h:inputText id="authorBirth" value="#{authorBean.author.authorBirth}"
                                     required="true"
requiredMessage="Ingrese fecha de nacimiento"
                                     styleClass="form-control"
>
<f:convertDateTime pattern="dd/MM/yyyy"/>
</h:inputText>
</td>
</tr>
<tr>
<td>
<label for="authorForm:authorNumber">Number </label>
<h:inputText id="authorNumber" value="#{authorBean.author.authorNumber}"
required="true" requiredMessage="Ingrese un número de teléfono válido"
styleClass="form-control"
>
<f:validator validatorId="sv.edu.udb.util.SVPhoneValidator" />
</h:inputText>
</td>
</tr>
<tr>

<td>

<label for="authorForm:genre">Genre</label>
<h:selectOneMenu id="genre" value="#{authorBean.author.literaryGenre}"
styleClass="form-control">
```

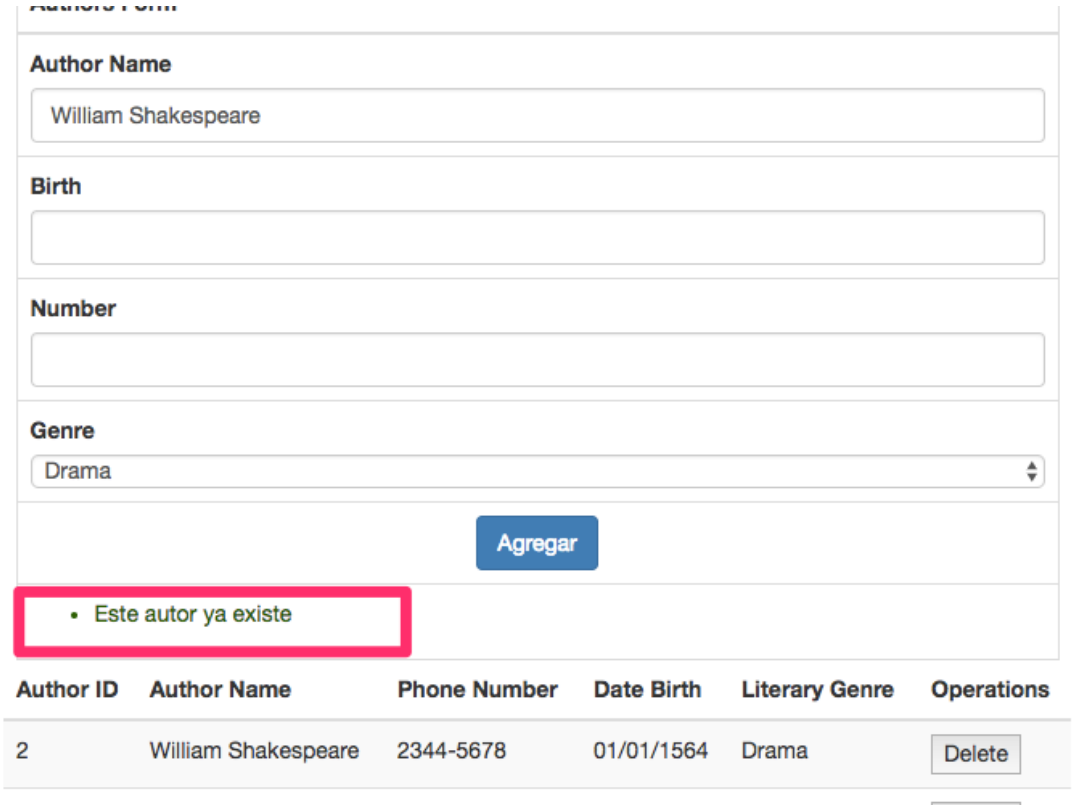
```
<f:selectItems value="#{authorBean.literaryGenreModel.literaryGenre}"
                                                var="l" itemLabel="#{l.name}"
itemValue="#{l.id}"

                                                />
</h:selectOneMenu>
</td>
</tr>
<tr>
<td>
<h:commandButton value="Agregar" action="#{authorBean.addAuthor()}"
                                               styleClass="btn btn-primary
center-block"
>
<f:ajax execute="@form"
                                      render=":authorsTable :authorForm
:datatableauthors"
                                      resetValues="true"/>

</h:commandButton>
</td>
</tr>
</tbody>
<tfoot>
<tr>
<td>
<h:messages id="successMessage" infoStyle="color:darkgreen"
errorStyle="color:darkred" />
<h:messages id="errorMessage" infoStyle="color:darkgreen"
globalOnly="true" errorStyle="color:darkred" />

</td>
</tr>
</tfoot>
</table>
</h:form>

</div>
<h:form id="datatableauthors">
<h:dataTable id="authorsTable" value="#{authorBean.authorModel.authors}"
var="c"
                     styleClass="table table-striped"
>
<h:column>
<f:facet name="header">
                     Author ID
</f:facet>
                #{c.authorId}
</h:column>
<h:column>
<f:facet name="header">
                     Author Name
</f:facet>
                #{c.authorName}
</h:column>

<h:column>
```

```
<f:facet name="header">
                    Phone Number
</f:facet>
                #{c.authorNumber}
</h:column>


<h:column>
<f:facet name="header">
                    Date Birth
</f:facet>
<h:outputText value="#{c.authorBirth}">
<f:convertDateTime pattern="dd/MM/yyyy"/>
</h:outputText>
</h:column>

<h:column>
<f:facet name="header">
                    Literary Genre
</f:facet>
                #{c.literaryGenre}
</h:column>
<h:column>
<f:facet name="header">
                    Operations
</f:facet>

<h:commandButton action="#{authorBean.deleteAuthor(c)}"
                                 onclick="if (! confirm('Do you want to
delete this author?')) return false"
                                 value="Delete">
<f:ajax execute="@form"  render="@form" />
</h:commandButton>
</h:column>

</h:dataTable>
</h:form>
</h:body>
</html>
```

Realice algunas operaciones con la página, notará que muchos componentes se actualizan sin refrescar la página



Intente ingresar un autor repetido, aunque no le impide agregarlo, le indica que ya ha sido agregado previamente.



IV. Ejercicios complementarios

Implemente la operación update para el formulario anterior.

Seleccione 2 controles de JSF de su preferencia y realice dos operaciones sobre la base de datos **utilizando AJAX**.

Por ejemplo:

Una aplicación que filtre los Autores por género al seleccionar un h:selectOneMenu y de manera asíncrona llene una tabla con los datos.

Contar el número de autores con presionar un botón y renderizarlo directamente en un h:outputText.