

PARTE 3 PROYECTO CASAS RURALES

JavaServer Faces

Autores: Iñigo, Julen y Adrián

Tiempo de realización: 17 horas

MEJORA

La mejora respecto a la entrega anterior ha sido mejorar la estética de la página cambiando el tamaño y el posicionamiento de los botones, y poniendo un fondo de color más agradable, además de añadir imágenes y explicar el correcto funcionamiento de la aplicación al final de este documento.

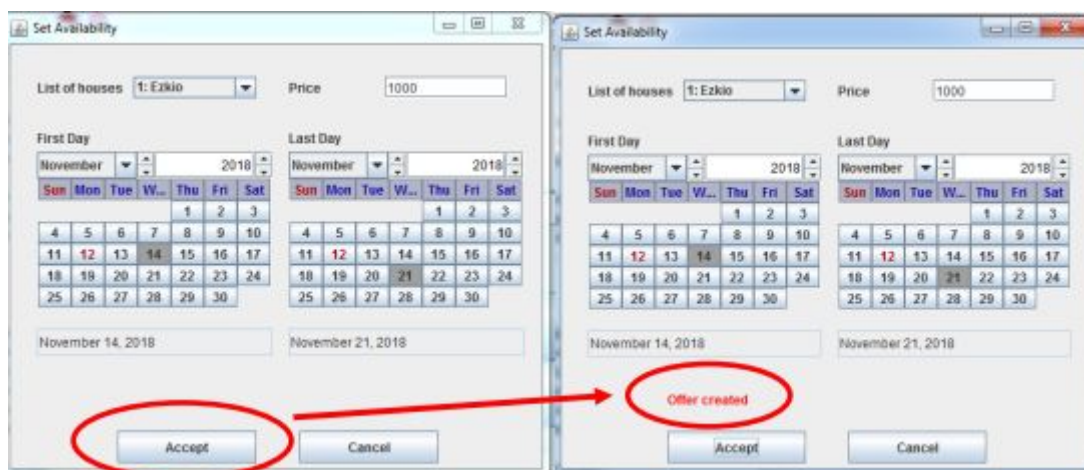
- **Instalación y comprobación de la aplicación pre-existente.**

Cogemos el proyecto LoginJSF que teníamos y copiamos el proyecto RuralHouseWS en la misma carpeta que este en nuestro workspace.

Probamos el correcto funcionamiento de este. Para ello ejecutaremos el caso de uso “Set availability” y “Query availability” con los datos que nos proporciona el enunciado.

Para el set creamos dos ofertas:

- 1:Ezkio, 14 noviembre, 21 noviembre, 1000
- 1:Ezkio, 26 noviembre, 30 noviembre, 200



Para el query comprobamos si la oferta fue creada correctamente y nos la muestra en la tabla creada. Para ello hacemos una búsqueda de la siguiente oferta:

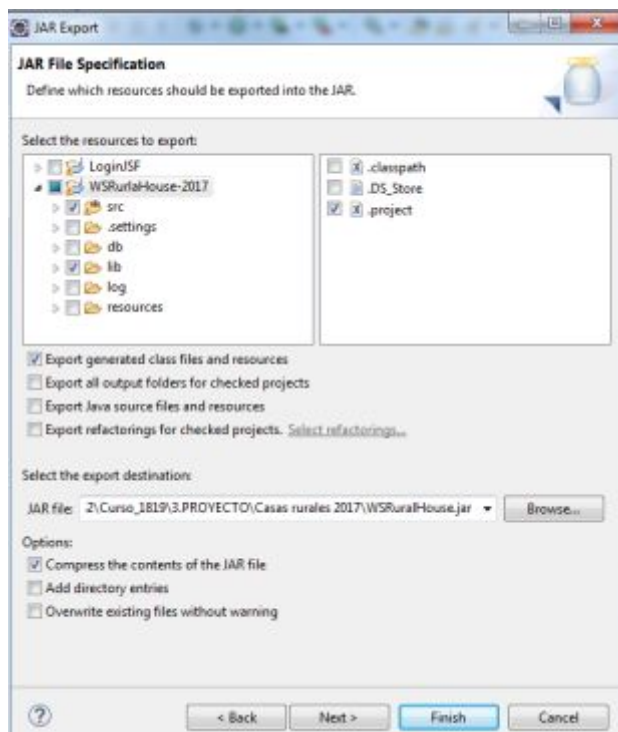
- 1:Ezkio, 14 noviembre, 20 días

Por tanto nos mostrará la primera oferta que habíamos creado.

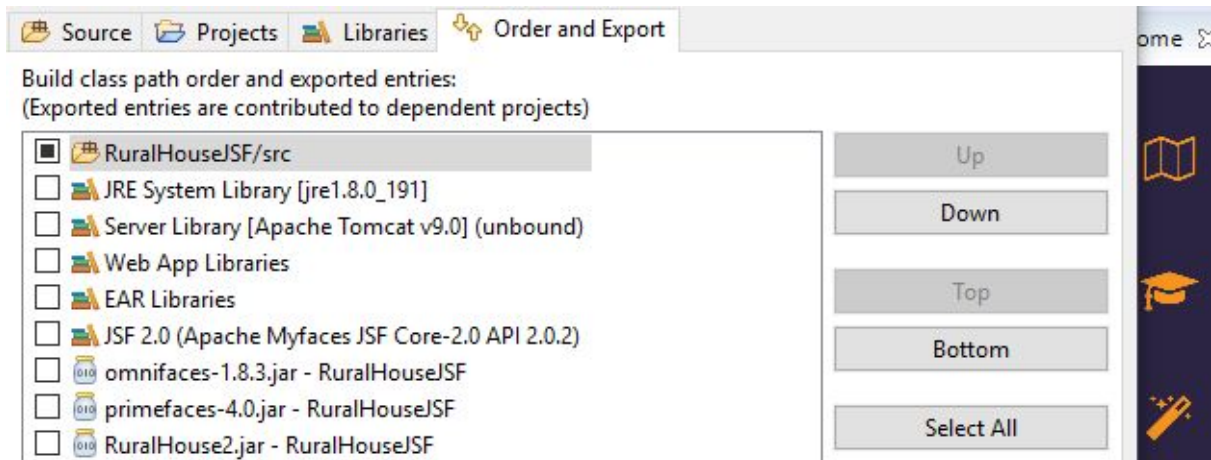


- **Creación de un nuevo proyecto JSF que utilice la lógica de negocio del proyecto que tenemos.**

Para esto, vamos a crear un nuevo proyecto llamado RuralHouseJSF. Ahora nos situamos en el proyecto RuralHouse que teníamos y lo exportamos como JAR incluyendo las librerías. Una vez hecho esto introducimos este archivo .jar en el directorio donde hemos creado nuestro proyecto de las casas JSF.



Ahora añadimos el .jar y las librerías necesarias al proyecto que estamos creando. Para ello nos situamos en el Java Build Path y añadimos:

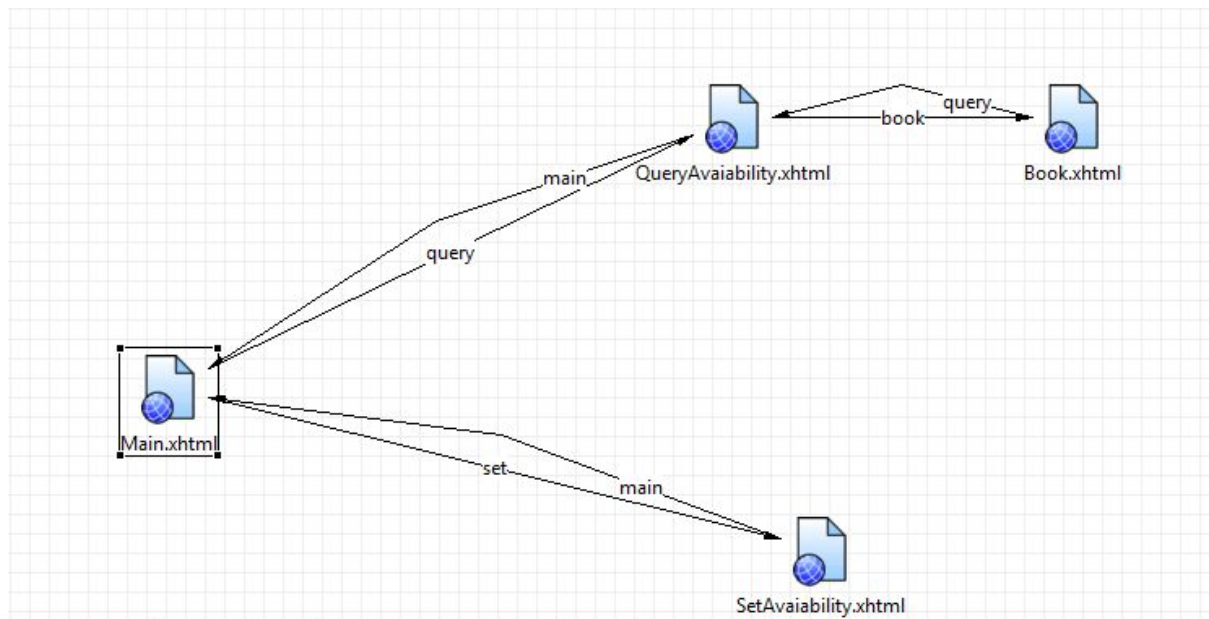


Por último, y antes de comenzar con la implementación, haremos accesibles las librerías para que el servidor web ejecute correctamente nuestra aplicación. Para ello las debemos añadir en el Deployment Assembly de nuestro proyecto:

Source	Deploy Path	
/src	WEB-INF/classes	
/WebContent	/	
jcalendar.jar - \RuralHouse\	WEB-INF/lib	Add...
JSF 2.0 (Apache Myfaces JSF Core-2.0 API 2.0.2)	WEB-INF/lib	Edit...
objectdb.jar - \RuralHouse\	WEB-INF/lib	Remove
omnifaces-1.8.3.jar - \RuralHouse\	WEB-INF/lib	
primefaces-4.0.jar - \RuralHouse\	WEB-INF/lib	
RuralHouse2.jar - \RuralHouse\	WEB-INF/lib	

- **Creación de las vistas JSF, beans y configuración necesaria para la aplicación.**

Lo primero que haremos será la configuración para la navegación entre páginas. Para ello crearemos (faces-config.xml -> Navigation Rule) un Main.xhtml que será la página principal, un QueryAvailability.xhtml y SetAvailability.xhtml que serán las interfaces para la búsqueda y creación de ofertas, y un Book.xhtml que actuará como la interfaz encargada de alquilar las ofertas.



Definiremos sus respectivos beans y los scopes con los que actuarán. En nuestro caso el scope más práctico para realizar las pruebas y el que hemos definido ha sido “session”.

Managed Bean Elements

The following managed beans are defined

- application
- session
 - queryBean
 - mainBean
 - setBean
- request

Managed Bean

This section describes general configuration of this managed bean

Managed Bean name*: queryBean

Managed Bean class*: bean.QueryBean

Managed Bean scope*: session

Initialization

You can initialize the managed bean's properties or itself if it is a subclass of java.util.Map or

Para la redirección entre las páginas hemos utilizado “<h:commandButton/>” para definirlos, de manera en que el action pasará el valor definido en el faces-config.xml.

Main.xhtml

```

<h:commandButton value="QueryAvailability" immediate="true" action="query"/>
<h:commandButton value="SetAvailavbilty" immediate="true" action="set"/>

```

QueryAvailability.xhtml

```

<br />
<h:commandButton value="Buscar ofertas" action="#{queryBean.action}" update="tablaOfertas, mensaje" />
<h:commandButton disabled="#{queryBean.alquiler}"
    value="Ir a alquilar" action="book" />
<h:commandButton value="Volver" action="main" immediate="true" />

```

Book.xhtml

```

<p:commandButton value="Alquilar" action="#{queryBean.alquilerOferta}"></p:commandButton>
<h:commandButton value="Volver" action="main" immediate="true"></h:commandButton>

```

Utilizamos el action llamando al bean que llama a la función alquilerOferta, que lo que hace es obtener el número de teléfono que se ha metido en el campo del

telefono, y sabiendo la oferta que se ha seleccionado en la tabla de antes, actualizar la oferta añadiendo el numero de telefono a dicha oferta e inicializando las variables de nuevo a su estado inicial, por si el mismo usuario quisiera realizar otra oferta.

```
public void aceptarAlquilerOferta() {
    boolean a = facade.createBook(SelectedOffer, telefono);
    if (a) {
        this.setAlquiler(false);
        this.setDiaFin(null);
        this.diaInicio=null;
        this.offers = new Vector<Offer>();
        this.setTelefono("");
        this.setNoches(0);
        this.setResultado("");
        this.setResultadoAlquiler("");
        this.setSelectedOffer(null);
        mensajeAlquiler = "Alquilada";
    }
    else {
        mensajeAlquiler = "La oferta no ha podido ser creada";
    }
}
```

Ahora vamos a proceder con la implementación de las interfaces y sus funcionalidades. Lo primero de todo será importar las librerías que necesitemos, omnifaces, primefaces, jcalendar ... Lo deberemos hacer tanto en el Java Build Path del proyecto como en los archivos .xhtml que las utilicen.

	<code><html xmlns="http://www.w3.org/1999/xhtml"</code>
<code>omnifaces-1.8.3.jar - RuralHouse</code>	<code>xmlns:ui="http://java.sun.com/jsf/facelets"</code>
<code>primefaces-4.0.jar - RuralHouse</code>	<code>xmlns:h="http://java.sun.com/jsf/html"</code>
<code>RuralHouse2.jar - RuralHouseJSF</code>	<code>xmlns:f="http://java.sun.com/jsf/core"</code>
<code>objectdb.jar - \RuralHouse\libs (</code>	<code>xmlns:o="http://omnifaces.org/ui"</code>
<code>jcalendar.jar - \RuralHouse\libs (</code>	<code>xmlns:of="http://omnifaces.org/functions"</code>
	<code>xmlns:p="http://primefaces.org/ui"></code>

El **main.xhtml** quedará de esta forma, ya que su única función será redireccionar a las funcionalidades especificadas.

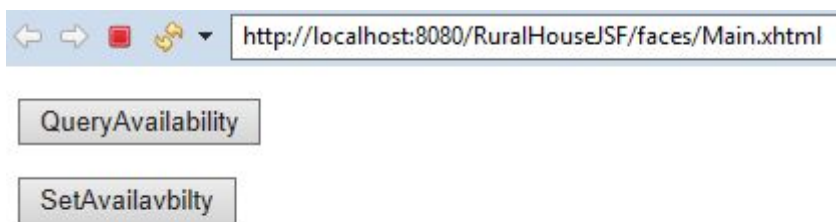
Esta clase no utilizará el bean ya que no utiliza funcionalidades internas.


```

<!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:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
<f:view>
<h:head>
<title>Elige</title>
</h:head>
<h:body>
<h:form>
<p>
<h:commandButton value="QueryAvailability" immediate="true" action="query"/>
</p>
<p>
<h:commandButton value="SetAvailavbilty" immediate="true" action="set"/>
</p>
</h:form>
</h:body>
</f:view>
</html>

```

Esta es la visualización de la interfaz gráfica de nuestro main.xhtml.



La clase **queryAvailability.xhtml** es más compleja, ya que tiene que obtener datos y utilizarlos, por tanto deberá utilizar su bean asociado. Voy a explicar esta clase por funcionalidades para que quede más claro.

Utilizaremos los import visualizados en una imagen anterior para poder utilizar un conversor, calendario ...

Lo primero que hacemos es crear una lista desplegable para que se muestren todas las casas. El valor de la etiqueta "selectOneMenu" obtiene una casa que será la que escojamos en el desplegable, mientras que "selectedItems" será un vector con todas las casas que mostraremos.

Para obtener las casas instanciaremos Facade en nuestro bean y llamaremos al método getAllRuralHouses() para que obtenga las casas de la base de datos.

Debemos utilizar el convertidor de omnifaces ya que de no utilizarlo no tendríamos forma de visualizar una casa. También se debe decir que son necesarios los métodos equals(), toString() y hashCode() para que el conversor pueda actuar de una manera correcta sobre las RuralHouses.

```

<h:selectOneMenu id="house" value="#{queryBean.casa}"
    converter="omnifaces.SelectItemsConverter">
    <f:selectItems value="#{queryBean.casas}" />
</h:selectOneMenu>

```

```

@Override
public String toString() {
    return this.houseNumber + ": " + this.city;
}
@Override
public boolean equals(Object obj) {
    RuralHouse other = (RuralHouse) obj;
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    if (!houseNumber.equals(other.houseNumber))
        return false;
    return true;
}

```

Implementación de método toString de la clase RuralHouse para la correcta visualización de esta.

En nuestro queryBean.java tendremos la instanciación del Facade, la lista de casas que se mostrarán y la casa escogida por el usuario. Por último creamos los getters y setters de estas dos variables a utilizar.

```

private FacadeImplementationWS facade = FacadeBean.getFacade();
public List<RuralHouse> casas = facade.getAllRuralHouses();
public RuralHouse casa;

public List<RuralHouse> getCasas() {
    return casas;
}

public void setCasas(Vector<RuralHouse> casas) {
    this.casas = casas;
}

public RuralHouse getCasa() {
    return casa;
}

public void setCasa(RuralHouse nombre) {
    this.casa = nombre;
}

```

Lo segundo que haremos será crear un calendario. Para ello utilizaremos la etiqueta “p:calendar”, en la que el valor utilizado será la fecha escogida que se guardará en la variable dialInicio de nuestro bean. El calendario incluye más funcionalidades para su visualización como popup o navigator.

Hemos introducido un mensaje de prueba que se mostrará cuando intentemos cambiar de página sin haber seleccionado una fecha. También hemos añadido una funcionalidad de ajax, mediante la cual se actualizará un texto (etiqueta “<h:outputText/>”) que mostrará en tiempo real la fecha que hayamos seleccionado. Para ello mediante la funcionalidad update se irá cambiando el valor de la etiqueta output. Se conseguirá una correcta visualización de la fecha añadiendo otra etiqueta

al output de la fecha añadida para realizar una conversión, `<f:convertDateTime/>` en la que se le añadirá el patrón que seguirá la fecha y la zona horaria de esta.

```
<p:calendar id="arrDay" value="#{queryBean.diaInicio}"
  navigator="true" mode="popup" required="true"
  requiredMessage="Debes seleccionar un dia de entrada en el calendario">
  <p:ajax event="dateSelect" update="diaInicio" />
</p:calendar>
<h:outputText id="diaInicio" value="#{queryBean.diaInicio}" style="color:blue">
  <f:convertDateTime pattern="dd/MM/yyyy" timeZone="CET" />
</h:outputText>
```

En cuanto a la funcionalidad del calendario relacionada con nuestro bean, simplemente necesitaremos una variable para identificar la fecha y su respectivo getter y setter.

```
public Date getDiaInicio() {
    return diaInicio;
}

public void setDiaInicio(Date diaInicio) {
    this.diaInicio = diaInicio;
}

private Date diaInicio;
```

Lo tercero será crear un input en el que introduciremos un número de noches para que se muestren las ofertas en ese rango de tiempo desde el día de inicio que hayamos seleccionado y las noches que introduzcamos.

Para esto, utilizaremos una variable noches mediante una etiqueta input que recogerá este valor en nuestro bean y se validará su correcto formato mediante la etiqueta `<f:validateLongRange>`, en el que se ha definido que como mínimo se deba introducir una noche. Utilizamos un conversor para enviar un correcto formato de nuestro valor introducido al bean.

También he de decir que hemos añadido una etiqueta `<h:message/>` para la variable noches por si el validador o el conversor diera cualquier error se mostrase por pantalla.

```
<h:inputText id="noches" value="#{queryBean.noches}">
  <f:validateLongRange minimum="1" />
  <f:converter converterId="javax.faces.Integer" />
</h:inputText>
<h:message for="noches" style="color:red" />
<br />
```

En el bean simplemente hemos añadido una variable noches con sus respectivos setter y getter.

```
public int getNoches() {
    return noches;
}

public void setNoches(int noches) {
    this.noches = noches;
}

private int noches;
```

Por último hemos creado la tabla para la visualización de las ofertas. Para ello hemos creado una etiqueta `<p:dataTable/>` que obtendrá el valor de las ofertas

(value:queryBean.offers), introduciremos el tipo de dato (var:offer) para saber que tipo de datos manejará la tabla. También se le informará de que las ofertas solo podrán ser seleccionadas individualmente(selectionMode:single), crearemos otra variable en el bean para poder obtener la oferta seleccionada (selection:queryBean.selectedOffer) y por último añadiremos el identificador para las ofertas (rowKey: offer.offerNumber) que accederá directamente al tipo de dato oferta.

```
1
2<table value="#{queryBean.offers}" id="tablaOfertas" var="offer" selectionMode="single" selection="#{queryAvailability.selectedOffer}" rowKey="#{offer.o
```

En cuanto a la visualización de los datos, mediante la etiqueta <p:column> crearemos las columnas para sus correspondientes tipos de datos y mediante las etiquetas <h:outputText> los visualizaremos accediendo a las ofertas obtenidas para la tabla.

En algunos de los casos serán necesarios utilizar algunos conversores para su correcta visualización, como podrían ser las fechas.

```
<p:column headerText="Number">
    <h:outputText value="#{offer.offerNumber}"></h:outputText>
</p:column>
<p:column headerText="Rural House">
    <h:outputText value="#{offer.ruralHouse}"></h:outputText>
</p:column>
<p:column headerText="First day">
    <h:outputText value="#{offer.firstDay}">
        <f:convertDateTime timeZone="CET" pattern="dd/MM/yyyy" />
    </h:outputText>
</p:column>
<p:column headerText="Last day">
    <h:outputText value="#{offer.lastDay}">
        <f:convertDateTime timeZone="CET" pattern="dd/MM/yyyy" />
    </h:outputText>
</p:column>
<p:column headerText="Price">
    <h:outputText value="#{offer.price}"></h:outputText>
</p:column>
```

Por último, se utilizan unas funciones ajax para la selección y deselección de los datos en tiempo real.

```
<p:ajax event="rowSelect"/>
<p:ajax event="rowUnselect"/>
</p:dataTable>
<br />
</p>
```

Ahora veremos cómo hemos estructurado nuestro bean para la obtención de las ofertas con offers y selectedOffer y sus respectivos setters y getters. Todos los demás atributos utilizados pertenecen a la clase offer.

```
private Offer selectedOffer;

private List<Offer> offers = new Vector<Offer>();
```

```

public Offer getSelectedOffer() {
    return SelectedOffer;
}

public void setSelectedOffer(Offer selectedOffer) {
    SelectedOffer = selectedOffer;
}

public List<Offer> getOffers() {
    return offers;
}

public void setOffers(Vector<Offer> offers) {
    this.offers = offers;
}

```

Por último, introducimos dos `outputText` que mostrarán dos mensajes alojados en las variables `resultado` y `resultadoAlquiler` del bean y que mostrarán unos `String` cuando se presione el botón para mostrar las ofertas explicado a continuación.

```

<h:outputText id="resultado" value="#{queryBean.resultado}" />
<br />
<h:outputText id="resultadoAlquiler"
    value="#{queryBean.resultadoAlquiler}" style="color:red" />

```

Para estas dos variables se utilizarán dos strings y sus respectivos getters y setters.

```

private String resultado= "";
private String resultadoAlquiler = "";

public String getResultadoAlquiler() {
    return resultadoAlquiler;
}

public void setResultadoAlquiler(String resultadoAlquiler) {
    this.resultadoAlquiler = resultadoAlquiler;
}

public String getResultado() {
    return resultado;
}

public void setResultado(String resultado) {
    this.resultado = resultado;
}

```

Por último tenemos tres "`<h:commandButton/>`". Los dos últimos sirven para la redirección entre páginas explicados al inicio del documento.

El primero es el botón encargado de hacer la búsqueda de las ofertas con los datos introducidos en la interfaz y el encargado de asignar los valores a las variables para que se muestren en la tabla.

Mediante el atributo `update` de la etiqueta se actualizará la tabla con los valores asignados en el método `action()` y se actualizarán los mensajes a mostrar asignando valores a las variables definidas mediante los `output`.

```

<h:commandButton value="Buscar ofertas" action="#{queryBean.action()}" update="tablaOfertas, mensaje" />
<h:commandButton disabled="#{queryBean.alquiler}" value="Ir a alquilar" action="book" />
<h:commandButton value="Volver" action="main" immediate="true" />

```

El método `queryBean.action` se encargará de realizar una instancia del calendario y añadir los valores introducidos en la interfaz a las variables `diaInicio` y `diaFin` (esta última mediante una operación del calendario para calcular el último día teniendo en cuenta el número de noches introducido). Obtendrá las ofertas para la casa introducida en el desplegable y las fechas (facade inicializado para toda la clase) y realizará un control de errores para estas ofertas mediante un `try catch`.

Por último dará los valores pertinentes a las variables encargadas de mostrar los mensajes (como he explicado en el punto anterior) y se encargará de modificar el valor de la variable "alquiler" que nos servirá al pulsar el botón "Ir a alquilar" para saber si se debe redireccionar a la siguiente página o no.

```

public void action() {
    try {
        Calendar c = Calendar.getInstance();
        //ApplicationFacadeInterfaceWS facade1 = BD.orderBD("Local");
        c.setTime(diaInicio);
        c.add(Calendar.DAY_OF_YEAR, noches);
        diaFin = c.getTime();
        offers = facade.getOffers(casa, diaInicio, diaFin);
        if (offers.isEmpty()){
            resultado = "There are no offers at these dates";
            alquiler = true;
        }
        else{
            resultado = "Select an offer if you want to book";
            alquiler = false;
        }
    } catch (Exception e) {
        System.out.println("No hay ofertas");
    }
}

```

El botón "ir a Alquilar" mirará si hay una oferta seleccionada correctamente, y en caso de ser correcto devolverá un valor definido en el `faces-config.xml` que nos permitirá redireccionar correctamente a la siguiente página. También modificará un mensaje de error que se mostrará en uno de los `<h:outputText/>` que hemos mencionado anteriormente.

```

public String alquilerOferta() {
    if (SelectedOffer==null){
        resultadoAlquiler = "You must select an offer to rent it";
        return "";
    }
    else{
        resultadoAlquiler = "";
        return "rent";
    }
}

```

Esta sería la interfaz gráfica definida para nuestro `QueryAvailability.xhtml`

http://localhost:8080/RuralHouseJSF/faces/Main.xhtml?sessionId=45CF1A179C45A160B85328D7A9EB2788

Rural House:

1: Ezkio

Selecciona la fecha de entrada:

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

House	First day	Last day	Price

Buscar ofertas Ir a alquilar Volver

Ahora voy a explicar cómo hemos definido nuestra página **book.xhtml**.

Poniéndonos en el caso de que hemos realizado un query de una oferta y hemos conseguido redirigirnos a la página book, las variables con la oferta seleccionada en el query estarán cargadas con los valores en el bean, lo que nos permitirá visualizarlos en esta página.

Por tanto, lo único que hemos hecho ha sido coger los valores de la casa, día inicio y día fin y mostrarlos por pantalla con un `<h:outputText/>`

```
<p>Rural House:</p>
<h:outputText value="#{queryBean.casa}"></h:outputText>
<p>Arrival day :</p>
<h:outputText value="#{queryBean.diaInicio}"></h:outputText>
<p>Departure day:</p>
<h:outputText value="#{queryBean.diaFin}"></h:outputText>
```

Introducimos una nueva variable teléfono mediante un `<h:inputText/>` que nos mostrará un mensaje en caso de no introducirlo.

```
<p>Introduce a telephone :</p>
<h:inputText value="#{queryBean.telefono}" required="true" requiredMessage="Necesitas introducir un numero de telefono"></h:inputText>
```

Para ello, hemos creado una variable teléfono en el bean con su correspondiente getter y setter.

```
private String telefono;

public String getTelefono() {
    return telefono;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}
```

Por último, definimos dos botones, uno para una simple redirección al main y otro para crear el alquiler de la oferta.

```
<h:commandButton value="Alquilar" action="#{queryBean.aceptarAlquilerOferta()}"></h:commandButton>
<h:commandButton value="Volver" action="main" immediate="true"></h:commandButton>
```

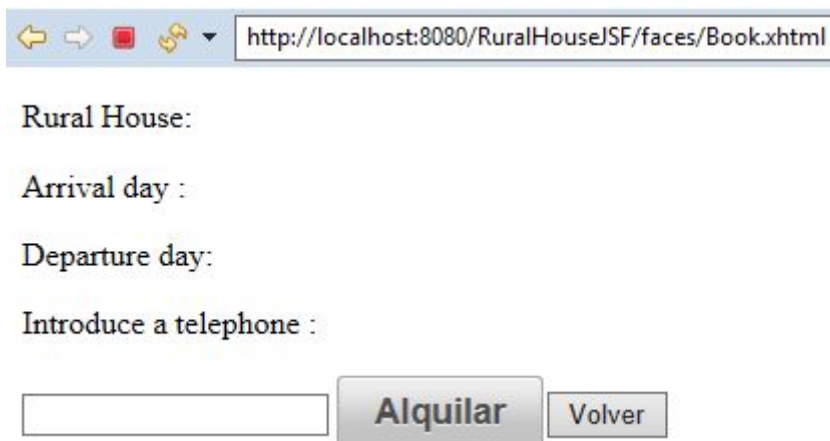

El botón volver devuelve el valor main definido en su action que redireccionará al main como está definido en el faces-config.xml.

El botón alquilar llamará a una función aceptarAlquilerOferta definida en el bean que se encargará de llamar al método createBook (facade instanciado para toda la clase) que creará el alquiler de la oferta con el teléfono y la oferta que le habíamos pasado desde el query.

Se realiza un control de errores para verificar la correcta creación de este alquiler, y en caso de ser correcto se ponen todas las variables relacionadas con el query y book a null para poder ejecutar correctamente de nuevo el query sin tener las variables inicializadas. Por último definimos el string definido en el faces-config.xml para avisar a la página de su redirección.

```
public void aceptarAlquilerOferta() {
    boolean a = facade.createBook(SelectedOffer, telefono);
    if (a) {
        this.setAlquiler(false);
        this.setDiaFin(null);
        this.diaInicio=null;
        this.offers = new Vector<Offer>();
        this.setTelefono("");
        this.setNoches(0);
        this.setResultado("");
        this.setResultadoAlquiler("");
        this.setSelectedOffer(null);
        mensajeAlquiler = "Alquilada";
    }
    else {
        mensajeAlquiler = "La oferta no ha podido ser creada";
    }
}
```

Está será la visualización de nuestra interfaz gráfica de la página book.xhtml.



Rural House:

Arrival day :

Departure day:

Introduce a telephone :

Alquilar **Volver**

La clase **setAvailability.xhtml** es parecida a queryAvailability, practicamente lo unico que cambia es la clase action, que es la siguiente:

```
public String action() {
    String resultado= "";
    try {
        Offer o = facade.createOffer(casa, diaInicio, diaFin, precio);
        if (o==null)
            resultado = (ResourceBundle.getBundle("Etiquetas").getString("BadDatesOverlap"));
        else resultado = (ResourceBundle.getBundle("Etiquetas").getString("OfferCreated"));

    } catch (java.lang.NumberFormatException e1) {
        //resultado = (jTextField3.getText()+ " " + ResourceBundle.getBundle("Etiquetas").getString("PriceNotValid"));
    } catch (OverlappingOfferExists e1) {
        resultado = (ResourceBundle.getBundle("Etiquetas").getString("OverlappingOffer"));
    }
    catch (BadDates e1) {
        resultado = (ResourceBundle.getBundle("Etiquetas").getString("LastDayBefore"));
    } catch (Exception e1) {
        e1.printStackTrace();
    }
    return resultado;
}
```

Por lo demás, esta sería el resto de la clase *setBean.java*:

```

public class SetBean {

    private FacadeImplementationWS facade = FacadeBean.getFacade();
    public List<RuralHouse> casas = facade.getAllRuralHouses();
    public RuralHouse casa;
    private float precio;
    private Date diaInicio;
    private Date diaFin;

    public List<RuralHouse> getCasas() {
        return casas;
    }
    public RuralHouse getCasa() {
        return casa;
    }

    public void setCasa(RuralHouse nombre) {
        this.casa = nombre;
    }
    public void setPrecio(float precio) {
        this.precio = precio;
    }
    public float getPrecio() {
        return this.precio;
    }
    public Date getDiaInicio() {
        return diaInicio;
    }

    public void setDiaInicio(Date diaInicio) {
        this.diaInicio = diaInicio;
    }
    public Date getDiaFin() {
        return diaFin;
    }
    public void setDiaFin(Date diaFin) {
        this.diaFin = diaFin;
    }
}

```

Este seria el SetAvailability.xhtml:

```

<title>Set Availability</title>
</h:head>
<h:body style="height: 291px; ">
<h:form>

    <p style="width: 143px;">List of houses:<h:selectOneMenu id="house"
value="#{setBean.casa}" converter="omnifaces.SelectItemsConverter" style="width: 103px; ">
    <f:selectItems value="#{setBean.casas}" />
</h:selectOneMenu></p>

    <h:message for="house"/>

    <p>Price</p>
    <h:inputText id="precio" value="#{setBean.precio}">
        <f:validateLongRange minimum="1" />
        <f:converter converterId="javax.faces.Float" />
    </h:inputText>
    <br /><h:message for="precio" style="color:red" />

    <p>First Day:</p>
    <p:calendar id="firstDay" value="#{setBean.diaInicio}"
        navigator="true" mode="popup" required="true"
        requiredMessage="Debes seleccionar un dia de entrada en el calendario">
        <p:ajax event="dateSelect" update="diaInicio" />
    </p:calendar>
    <h:outputText id="diaInicio" value="#{setBean.diaInicio}" style="color:blue">
        <f:convertDateTime pattern="dd/MM/yyyy" timeZone="CET" />
    </h:outputText>

    <p>Last Day:</p>
    <p:calendar id="lastDay" value="#{setBean.diaFin}"
        navigator="true" mode="popup" required="true"
        requiredMessage="Debes seleccionar un dia de salida en el calendario">
        <p:ajax event="dateSelect" update="diaFin" />
    </p:calendar>
    <h:outputText id="diaFin" value="#{setBean.diaFin}" style="color:blue">
        <f:convertDateTime pattern="dd/MM/yyyy" timeZone="CET" />
    </h:outputText>
    <br />
    <br />
    <p>
    <h:commandButton value="Accept" action="#{setBean.action}" />
    <h:commandButton value="Volver" action="main" immediate="true" />
    </p>
</h:form>
</h:body>
</f:view>
</html>

```

Y este sería el resultado:

List of houses:
3: Bilbo

Price
30

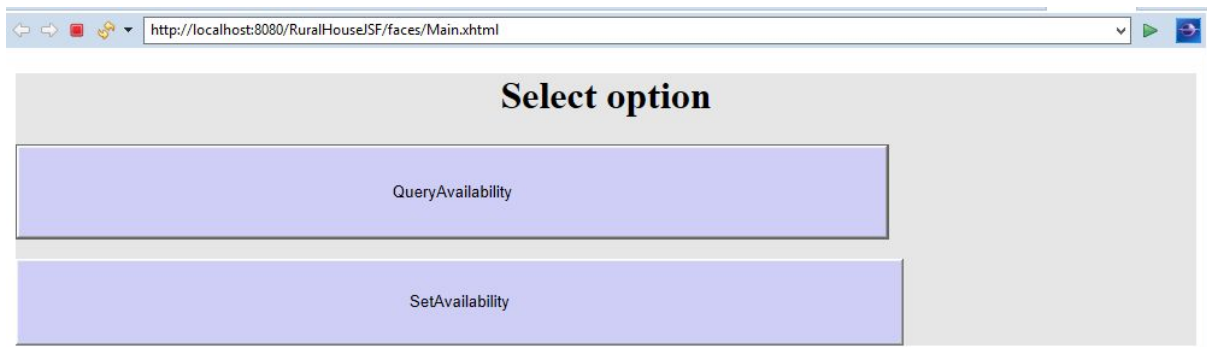
First Day:
26/11/18 26/11/2018

Last Day:
29/11/18 29/11/2018

Accept Volver

FUNCIONAMIENTO DE LA APLICACIÓN

Al iniciar la aplicación nos salen los dos siguientes botones que nos llevarán a sus respectivas ventanas.



Al pulsar el botón de SetAvailability nos llevará a su respectiva ventana y probaremos a crear dos ofertas para la casa rural de Ezkio.

A screenshot of a web browser window showing the 'SET AVAILABILITY' form. The browser's address bar displays 'http://localhost:8080/RuralHouseJSF/faces/SetAvailability.xhtml'. The page has a light gray background and a title 'SET AVAILABILITY' in bold black text. Below the title, there is a form with the following fields: 'List of houses:' with a dropdown menu showing '1: Ezkio'; 'Price' with a text input field containing '30.0'; 'First Day:' with a date input field containing '11/11/18' and a blue link '11/11/2018'; 'Last Day:' with a date input field containing '12/11/18' and a blue link '12/11/2018'. At the bottom of the form, there are two buttons: 'Accept' and 'Volver'. Below the buttons, there is a red text message 'Oferta creada'.

Una vez creada nos saldrá un mensaje de que la creación a ido bien.

http://localhost:8080/RuralHouseJSF/faces/SetAvailability.xhtml

SET AVAILABILITY

List of houses:
1: Ezkio

Price
40.0

First Day:
13/11/18 13/11/2018

Last Day:
14/11/18 14/11/2018

Oferta creada

Una vez tenemos las dos ofertas creadas para la casa de Ezkio, vamos al QueryAvailability, y seleccionando una fecha, y un número de noche nos saldrán en la tabla las ofertas disponibles para esa fecha.

http://localhost:8080/RuralHouseJSF/faces/Main.xhtml

QUERY AVAILABILITY

Rural House:
1: Ezkio

Selecciona la fecha de entrada:
10/11/18 10/11/2018

Number of nights:
8

Number	Rural House	First day	Last day	Price
No records found.				

http://localhost:8080/RuralHouseJSF/faces/QueryAvailability.xhtml

QUERY AVAILABILITY

Rural House:

1: Ezkio

Selecciona la fecha de entrada:

10/11/18 10/11/2018

Number of nights:

8

Number	Rural House	First day	Last day	Price
6	1: Ezkio	11/11/2018	12/11/2018	30.0
7	1: Ezkio	13/11/2018	14/11/2018	40.0

Select an offer if you want to book

Buscar ofertas Ir a alquilar Volver

Una vez nos salgan las dos ofertas en la tabla, pulsamos una de las dos ofertas y pulsamos el botón de Ir a alquilar, la cual nos llevara a la respectiva ventana para poder alquilarla.

http://localhost:8080/RuralHouseJSF/faces/QueryAvailability.xhtml

QUERY AVAILABILITY

Rural House:

1: Ezkio

Selecciona la fecha de entrada:

10/11/18 10/11/2018

Number of nights:

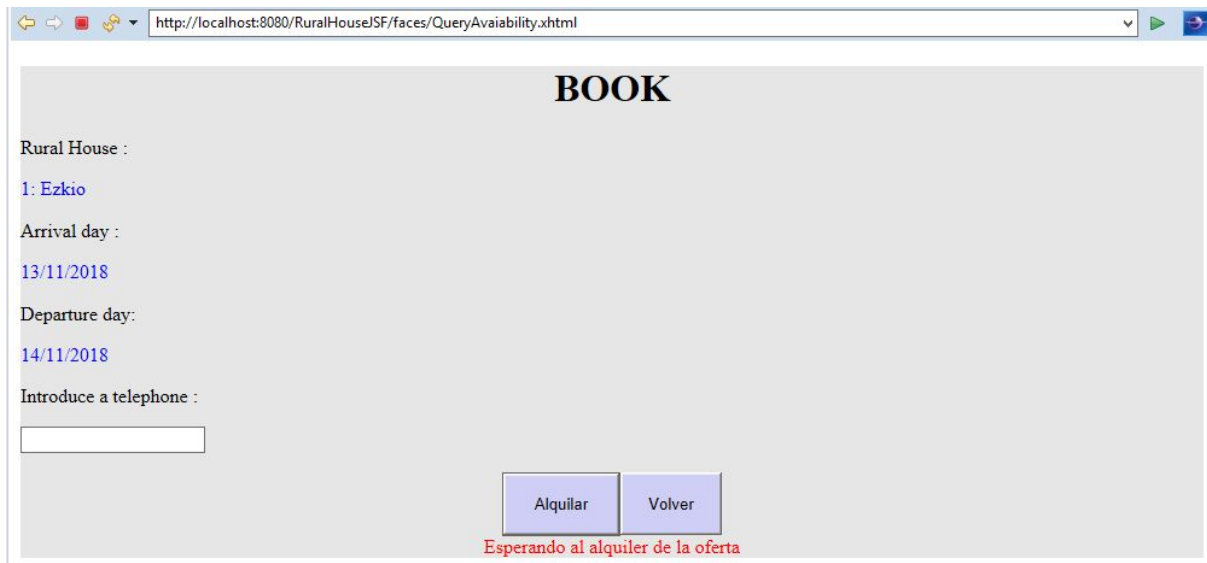
8

Number	Rural House	First day	Last day	Price
6	1: Ezkio	11/11/2018	12/11/2018	30.0
7	1: Ezkio	13/11/2018	14/11/2018	40.0

Select an offer if you want to book

Buscar ofertas Ir a alquilar Volver

En la ventana del Book nos saldrá los datos de la oferta y tendremos que introducir un número de telefono para realizar la reserva.

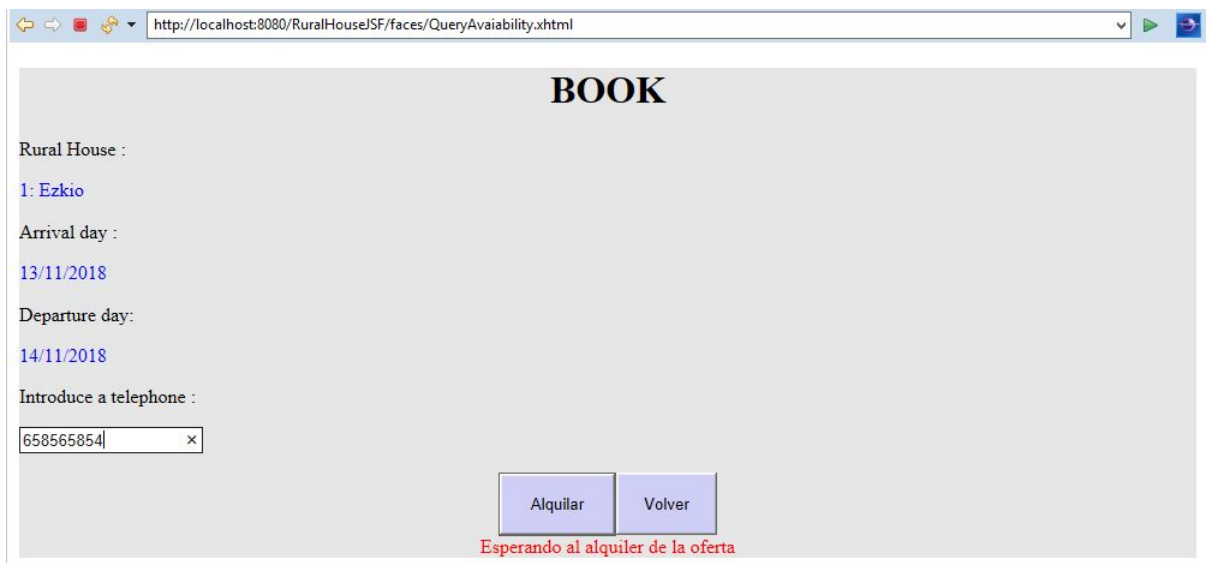


The screenshot shows a web browser window with the URL `http://localhost:8080/RuralHouseJSF/faces/QueryAvaibility.xhtml`. The page has a title **BOOK**. Below the title, the following information is displayed:

- Rural House :
1: Ezkio
- Arrival day :
13/11/2018
- Departure day:
14/11/2018
- Introduce a telephone :

At the bottom, there are two buttons: **Alquilar** and **Volver**. Below the buttons, the text **Esperando al alquiler de la oferta** is displayed in red.

Una vez introducido el numero de telefono y pulsemos el boton de Alquilar nos saldrá un mensaje de que la oferta se ha alquilado correctamente.



This screenshot is identical to the previous one, but the phone number `658565854` has been entered into the text field. The **Alquilar** button is now highlighted with a blue border, indicating it is the active element.

http://localhost:8080/RuralHouseJSF/faces/Book.xhtml

BOOK

Rural House :

1: Ezkio

Arrival day :

13/11/2018

Departure day:

14/11/2018

Introduce a telephone :

Alquilada

Y una vez alquilada una oferta no nos debería salir en la tabla al buscar las ofertas disponibles.

http://localhost:8080/RuralHouseJSF/faces/QueryAvailability.xhtml

QUERY AVAILABILITY

Rural House:

1: Ezkio

Selecciona la fecha de entrada:

10/11/18 10/11/2018

Number of nights:

9

Number	Rural House	First day	Last day	Price
6	1: Ezkio	11/11/2018	12/11/2018	30.0

Select an offer if you want to book