

**Tema**

TALLER WEB CON APACHE NETBEANS

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

13/01/2023

TALLER WEB CON APACHE NETBEANS

[1. INTRODUCCIÓN 3](#_Toc124273014)

[2. OBJETIVOS 3](#_Toc124273015)

[2.1. OBJETIVO GENERAL 3](#_Toc124273016)

[2.2. OBJETIVOS ESPECÍFICOS 3](#_Toc124273017)

[3. MARCO TEÓRICO 3](#_Toc124273018)

[3.1 PATRÓN MODELO VISTA CONTROLADOR 3](#_Toc124273019)

[3.1.1. MODELO 4](#_Toc124273020)

[3.1.2. VISTA 4](#_Toc124273021)

[3.1.3. CONTROLADOR 4](#_Toc124273022)

[3.2 Jakarta Persistence 4](#_Toc124273023)

[3.2.1. Jakarta y sus especificaciones 5](#_Toc124273024)

[3.3 Java Server Faces (JSF) 5](#_Toc124273025)

[3.3.1. Elementos de Java Server Faces 5](#_Toc124273026)

[3.3.2. Características de JSF 5](#_Toc124273027)

[3.3.3. Desventajas de JSF 6](#_Toc124273028)

[3.3.4. Etiquetas de Java Server Faces 6](#_Toc124273029)

[3.4 XHTML 8](#_Toc124273030)

[3.4.1. Ventajas de XHTML 8](#_Toc124273031)

[3.4.2. Desventajas de XHTML 9](#_Toc124273032)

[3.5 JSF CDI Bean 9](#_Toc124273033)

[3.6 Inyección de dependencias JSF 9](#_Toc124273034)

[3.7 beans.xml (CDI configuration File) 10](#_Toc124273035)

[3.8 Class Validator 10](#_Toc124273036)

[4. PARTE PRÁCTICA 11](#_Toc124273037)

[4.1 CREACIÓN DEL PROYECTO 11](#_Toc124273038)

[4.2 ESTRUCTURACIÓN DEL PROYECTO 13](#_Toc124273039)

[4.3 CODIFICACIÓN DEL PROYECTO 14](#_Toc124273040)

[4.3.1. CODIFICACIÓN DEL MODELO 14](#_Toc124273041)

[4.3.2. CODIFICACIÓN DE LA VISTA 17](#_Toc124273042)

[4.3.3. CODIFCACIÓN DEL CONTROLADOR 19](#_Toc124273043)

[4.4 EJECUCIÓN DEL PROYECTO MANAGED BEANS 21](#_Toc124273044)

[5. CONCLUSIONES 25](#_Toc124273045)

[6. RECOMENDACIONES 25](#_Toc124273046)

[7. REFERENCIAS 26](#_Toc124273047)

**INDICE DE IMÁGENES**

[Figura 1. Arquitectura MVC 4](#_Toc124240703)

[Figura 2. Creación de un nuevo proyecto. 12](#_Toc124240704)

[Figura 3. Selección del tipo de proyecto a crear. 12](#_Toc124240705)

[Figura 4. Configuración de los parámetros del proyecto. 13](#_Toc124240706)

[Figura 5. Configuración del servidor y versión de JEE. 13](#_Toc124240707)

[Figura 6. Configuración de los Frameworks a utilizar. 14](#_Toc124240708)

[Figura 7. Creación de paquetes. 14](#_Toc124240709)

[Figura 8. Creación de los paquetes Modelo y Controlador. 14](#_Toc124240710)

[Figura 9. Creación del archivo en el paquete modelo. 15](#_Toc124240711)

[Figura 10. Selección del tipo de archivo. 15](#_Toc124240712)

[Figura 11. Configuración de parámetros del archivo. 16](#_Toc124240713)

[Figura 12. Creación de archivos para la Vista. 18](#_Toc124240714)

[Figura 13. Selección del tipo de archivo que se va a crear. 18](#_Toc124240715)

[Figura 14. Asignación de nombre al archivo. 19](#_Toc124240716)

[Figura 15. Creación del archivo para el paquete controlador. 20](#_Toc124240717)

[Figura 16. Configuración de parámetros del archivo a crear. 21](#_Toc124240718)

[Figura 17. Clean and Build del proyecto 22](#_Toc124240719)

[Figura 18. Pantalla principal de la aplicación. 22](#_Toc124240720)

[Figura 19. Llenar información solicitada. 22](#_Toc124240721)

[Figura 20. Ejecución de la aplicación. 23](#_Toc124240722)

[Figura 21. Creación de nuevo archivo para validar el correo. 23](#_Toc124240723)

[Figura 22. Selección del tipo de archivo. 24](#_Toc124240724)

[Figura 23. Configuración y asignación de nombre del archivo a crear. 24](#_Toc124240725)

[Figura 24. Agregar la validación al TAG. 26](#_Toc124240726)

[Figura 25. Comprobar validación de correo. 26](#_Toc124240727)

**ÍNDICE DE TABLAS**

[Tabla 1. Etiquetas básicas de JSF – HTML 6](#_Toc124240758)

[Tabla 2. Etiquetas Java Server Faces – CORE 8](#_Toc124240759)

[Tabla 3. Ejemplo de validador de longitud. 10](#_Toc124240760)

[Tabla 4. Ejemplo de validadores personalizados. 11](#_Toc124240761)

[Tabla 5. Implementación de la clase Validator. 11](#_Toc124240762)

[Tabla 6. Codificación del archivo RegistrationBean.java 15](#_Toc124240763)

[Tabla 7. Codificación del archivo confirmación.html 18](#_Toc124240764)

[Tabla 8. Codificación del archivo registrationBeanController.java 20](#_Toc124240765)

[Tabla 9. Codificación del archivo EmailValidator.java 24](#_Toc124240766)

# INTRODUCCIÓN

Las aplicaciones web deben estar desarrolladas bajo un patrón de arquitectura para que sea más sencillo corregir errores, implementar nuevas funcionalidades, realizar pruebas y mantenimiento de la aplicación. El patrón arquitectura más común es MVC (Modelo-Vista-Controlador), aunque existen otras variaciones más modernas como MVVM o MVP.

# OBJETIVOS

## OBJETIVO GENERAL

Analizar los conceptos y pasos para elaborar un formulario web implementado con el patrón de arquitectura MVC usando la plataforma Apache Netbeans y el marco de desarrollo, para entender cómo se organiza y comunica una aplicación web con MVC.

## OBJETIVOS ESPECÍFICOS

* Analizar la arquitectura Modelo-Vista-Controlador en Java Web Application para comprender como se debe organizar una aplicación web bajo esta arquitectura.
* Identificar los pasos para implementar un formulario web que registre información de un usuario.
* Documentar todo el proceso de análisis y diseño utilizando el IDE (Entorno de desarrollo integrado) Apache Netbeans

# MARCO TEÓRICO

## PATRÓN MODELO VISTA CONTROLADOR

Es un patrón de arquitectura para el desarrollo de aplicaciones software donde su objetivo es el separar la lógica del negocio de la interfaz de usuario lo cual lo hace escalable facilitando la evolución por separado de sus partes, incrementa la reutilización y flexibilidad.

Tuvo su origen en 1979 y es también conocido como patrón MVC, divide una aplicación software en tres grandes partes bien diferenciadas las cuales son Modelo, Vista y Controlador.

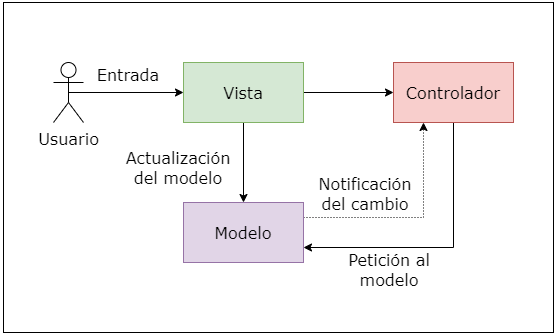


Figura . Arquitectura MVC

### **MODELO**

Es una capa donde se localiza la funcionalidad central y los datos, se comunica con el controlador y la base de datos.

Es la capa donde se trabaja con los datos, por tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

### **VISTA**

Es la capa donde se muestra la información al usuario donde es posible definir una o más vista de usuario en un software, se comunica con el controlador y el usuario.

Las vistas, como su nombre nos hacen entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generarán la salida, tal como nuestra aplicación requiera.

### **CONTROLADOR**

Es la capa donde se manejan las entradas del usuario se separa la representación interna de la información y la forma en la que se le muestra al usuario, se comunica con el modelo.

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad, es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

## Jakarta Persistence

Jakarta EE es la nueva plataforma de Java Enterprise Edition (Java EE), que ha llegado hasta la versión 8. Hasta ahora, todas las especificaciones de Java EE han sido fuertemente lideradas por Oracle, que es el propietario de Java y, por lo tanto, de Java EE como una extensión natural de este lenguaje. Sin embargo, cada vez es más necesario que Java EE se dirija hacia el mundo del código abierto para que los estándares sean más abiertos. Esto se debe a que la industria de la tecnología está cada vez más orientada hacia el código abierto, y es importante que Java EE se mantenga al día con estos cambios. Además, el enfoque en el código abierto permitirá que más desarrolladores contribuyan a la plataforma y mejoren su calidad y funcionalidad. En resumen, Jakarta EE es una nueva versión de Java EE que se enfoca en el mundo del código abierto y está diseñada para ser más accesible y relevante en la industria actual de la tecnología.

Hoy por hoy la versión que presumiblemente se adoptará es prácticamente idéntica a Java EE 8. Así pues, tendremos que esperar a Jakarta EE 9 para empezar a ver evoluciones importantes cómo puede ser las nuevas APIS de persistencia para soluciones asíncronas tipo BigData.

### **Jakarta y sus especificaciones**

Jakarta EE cuenta con una amplia variedad de especificaciones, algunas de las cuales son destacables:

* Jakarta Enterprise Java Beans: Uno de los proyectos más importantes, ya que ha sido muy utilizado por aquellos que siempre han necesitado usar los estándares de Java EE.
* Jakarta Persistence: Otro de los proyectos clave, ya que permite avanzar en la definición de la capa de persistencia y abordar nuevas funcionalidades cada vez más necesarias.
* Jakarta RESTFul Web Services: La especificación que se refiere a los servicios Web de tipo REST, que hoy en día es imprescindible en cualquier proyecto moderno.

En resumen, Jakarta EE tiene un amplio conjunto de especificaciones que cubren diferentes áreas y que son esenciales para el desarrollo de aplicaciones empresariales modernas.

## Java Server Faces (JSF)

Es un Framework el cual implementa una arquitectura MODELO VISTA CONTROLADOR, que se encuentra basado en componentes y eventos del lado del servidor donde mantiene una representación de la interfaz de usuario en el lado del servidor el cual es presentado al cliente.

### **Elementos de Java Server Faces**

JSF (Java Server Faces) es un marco de trabajo para el desarrollo de aplicaciones web basadas en Java. Algunos elementos clave de JSF son:

* Componentes: JSF proporciona un conjunto de componentes predefinidos que se pueden utilizar para construir la interfaz de usuario de una aplicación. Estos componentes incluyen elementos como formularios, botones, menús y tablas.
* Vistas: JSF permite la creación de vistas basadas en plantillas, que se pueden reutilizar en diferentes partes de la aplicación.
* Managed beans: Los managed beans son objetos Java que se utilizan para encapsular la lógica de negocio y los datos que se muestran en la interfaz de usuario.
* Validación y conversión de datos: JSF proporciona herramientas para validar y convertir los datos que se envían al servidor desde la interfaz de usuario.
* Integración con otras tecnologías: JSF se puede integrar con otras tecnologías de servidor, como EJBs (Enterprise Java Beans) y JPA (Java Persistence API), para proporcionar funcionalidades adicionales.

### **Características de JSF**

* Facilita la separación de responsabilidades entre la interfaz de usuario y la lógica de negocio.
* Proporciona un conjunto de componentes predefinidos que se pueden utilizar para construir la interfaz de usuario de una aplicación.
* Permite la creación de vistas basadas en plantillas, que se pueden reutilizar en diferentes partes de la aplicación.
* Proporciona herramientas para validar y convertir los datos que se envían al servidor desde la interfaz de usuario.
* Se puede integrar con otras tecnologías de servidor, como EJBs (Enterprise Java Beans) y JPA (Java Persistence API), para proporcionar funcionalidades adicionales.
* Facilita la creación de aplicaciones web escalables y de alta calidad gracias a su enfoque en la separación de responsabilidades y en la reutilización de componentes.

### **Desventajas de JSF**

* Complejidad: JSF puede ser un marco más complejo que otras opciones debido a su enfoque en la construcción de componentes reutilizables y a su enfoque en la separación de la lógica de la interfaz de usuario.
* Rendimiento: Debido a la complejidad de JSF, puede haber un impacto en el rendimiento de la aplicación en comparación con marcos más simples.
* Dificultad para aprender: Aunque JSF ofrece muchas ventajas, también puede ser difícil de aprender para algunos desarrolladores debido a su complejidad y a la cantidad de conceptos y tecnologías que deben entenderse.
* Dependencia de Java EE: JSF es parte de la plataforma Java EE, lo que significa que las aplicaciones que utilizan JSF dependen de la plataforma y están limitadas a ejecutarse en entornos que la soporten.
* Menor popularidad: Aunque JSF ha sido utilizado con éxito en muchas aplicaciones, no es tan popular como algunos otros marcos web, lo que puede dificultar encontrar recursos o profesionales experimentados en el marco.

### **Etiquetas de Java Server Faces**

Tabla 1. Etiquetas básicas de JSF – HTML

|  |  |  |
| --- | --- | --- |
| Tipo | Etiqueta | Descripción |
| Agrupación | <h:form /> | Define el formulario JSF – HTML. |
| <h:panelGrid /> | Crea una tabla con los componentes incluidos en el panelGrid |
| <h:panelGroup /> | Permite agrupar dos o más componentes |
| Entrada | <h:inputText /> | Campo de texto para entrada de datos. |
| <h:inputHidden /> | Vincula un campo oculto en el formulario. |
| <h:inputSecret /> | Vincula un campo de texto de datos de tipo contraseña (no muestra el contenido). |
| <h:inputTextarea /> | Componente área de texto (entrada de datos multilínea). |
| Salida | <h:outputText /> | Etiqueta de texto. |
| <h:outputLabel /> | Etiqueta de texto (se puede asociar a otro componente). |
| <h:outputLink /> | Permite crear un enlace o hipertexto. |
| <h:outputFormat /> | Permite dar formato a un texto de salida. |
| Errores y mensajes | <h:message /> | Permite imprimir un mensaje de error en la página. |
| <h:messages /> | Permite imprimir varios mensajes de error en la página. |
| Comandos, botones y enlaces | <h:commandButton /> | Crea un botón para realizar acciones. |
| <h:commandLink /> | Crea un enlace al que se le puede asociar a un botón. |
| Selección | <h:selectOneMenu /> | Lista de selección múltiple con única respuesta. |
| <h:selectOneRadio /> | Lista de radioButtons. |
| <h:selectManyListbox /> | Lista de selección múltiple con múltiple respuesta. |
| <h:selectManyCheckbox /> | Lista de checkbox. |
| <h:selectBooleanCheckbox /> | Componente checkbox. |
| <h:selectManyMenu /> | Componente de selección múltiple. |
| Tablas de datos | <h:dataTable /> | Permite vincular una tabla de datos dinámicos, se obtienen de una lista de elementos. |
| Imágenes | <h:graphicImage /> | Permite vincular imágenes. |

Tabla 2. Etiquetas Java Server Faces – CORE

|  |  |
| --- | --- |
| Etiqueta | Descripción |
| <f:view /> | Permite crear una vista en la página. |
| <f: subview /> | Permite crear una sub vista en la página. |
| <f: facet /> | Permite añadir un elemento facet a un componente (para cabeceras y pies de página por ejemplo). |
| <f: attribute /> | Añade un atributo a un componente mediante clave – valor. |
| <f: param /> | Añade un parámetro a un componente. |
| <f: actionListener /> | Añade una acción (listener) a un componente. |
| <f: valueChangeListener /> | Añade un nuevo valor al escuchador de un componente. |
| <f: convertDateTime /> | Agrega fecha a un componente. |
| <f: convertNumber /> | Añade conversión de un número a un componente |
| <f: validator /> | Añade un validador a un componente. |
| <f: validateDoubleRange /> | Permite validar un rango de tipo double para datos ingresados en el componente. |
| <f: validateLength /> | Permite validar longitud del valores de ingresados en un componente. |
| <f: validateLongRange /> | Permite validar un rango de tipo long para valores ingresados en componentes. |
| <f: loadBundle /> | Carga origen de un elemento bundle. |
| <f: selectitems /> | Permite especificar los ítems que se van a cargar en un componente de selección (tomados por lo regular de un arreglo o listas). |
| <f: selectitem /> | Permite especificar el ítem que se va a mostrar en un componente de selección (cuando se hace directamente). |
| <f: verbatim /> | Añade una marca a una página jsf. |

## XHTML

XHTML (eXtensible HyperText Markup Language) es un lenguaje de marcado para la representación de contenido en la World Wide Web. Es una variante de HTML (HyperText Markup Language) que sigue las reglas de sintaxis de XML (eXtensible Markup Language).

Los archivos XHTML son documentos de texto que utilizan etiquetas para describir y dar formato al contenido. Estas etiquetas son usadas para dividir el documento en diferentes secciones y para aplicar formato al texto, imágenes y otros elementos incluidos en el documento.

Los archivos XHTML se suelen utilizar para crear páginas web y a menudo se utilizan en conjunción con marcos web como JSF (JavaServer Faces) para crear aplicaciones web.

### **Ventajas de XHTML**

* Estricto: XHTML sigue las reglas de sintaxis de XML, lo que significa que es un lenguaje más estricto y menos permissivo que HTML. Esto puede hacer que el código sea más legible y fácil de mantener.
* Extensible: Al ser una versión extensible de HTML, XHTML permite la creación de nuevas etiquetas y atributos personalizados, lo que permite un mayor control sobre el contenido y el formato de la página.
* Mejor soporte para aplicaciones móviles: XHTML es más adecuado para el uso en aplicaciones móviles debido a su mayor estrictura y porque es más fácilmente procesado por dispositivos con capacidades limitadas de procesamiento.
* Mayor compatibilidad con los estándares: XHTML es más compatible con los estándares web debido a su adherencia a las reglas de sintaxis de XML.
* Mayor facilidad para integrar otros lenguajes: Al ser un lenguaje basado en XML, XHTML es más fácil de integrar con otros lenguajes de programación y tecnologías, como CSS (Cascading Style Sheets) y JavaScript.

### **Desventajas de XHTML**

* Mayor complejidad: XHTML puede ser más complejo que HTML debido a su mayor estructura y a la necesidad de seguir las reglas de sintaxis de XML.
* Mayor tiempo de desarrollo: Debido a su mayor complejidad, puede tomar más tiempo desarrollar y mantener una página web utilizando XHTML en comparación con HTML.
* Mayor tamaño de archivo: Los archivos XHTML suelen ser más grandes que los archivos HTML debido a la necesidad de cerrar todas las etiquetas y utilizar atributos en forma de pares clave/valor. Esto puede aumentar el tiempo de carga de la página.
* Mayor dificultad para convertir desde HTML: Si ya se tiene una página web existente escrita en HTML, puede ser difícil y requerir mucho trabajo convertirla a XHTML debido a la mayor estructura y complejidad del lenguaje.
* Mayor dificultad para encontrar recursos y profesionales: Aunque XHTML ha sido utilizado con éxito en muchas aplicaciones, no es tan popular como HTML, lo que puede dificultar encontrar recursos o profesionales experimentados en el lenguaje.

## JSF CDI Bean

JSF (JavaServer Faces) es un marco de aplicaciones web que proporciona un conjunto de componentes y herramientas para la construcción de interfaces de usuario para aplicaciones web. CDI (Contexts and Dependency Injection) es un marco de inyección de dependencias para Java que permite a los desarrolladores inyectar objetos en otras clases de forma transparente.

Un CDI Bean es un objeto que es manejado por el contenedor CDI y que puede ser inyectado en otras clases como una dependencia. Un CDI Bean puede ser utilizado en aplicaciones JSF para proporcionar funcionalidad y lógica de negocio en la interfaz de usuario.

Por ejemplo, un CDI Bean puede ser utilizado para recuperar datos de una base de datos y proporcionarlos a un componente de la interfaz de usuario de JSF, como una tabla o una lista desplegable. Los CDI Beans también pueden ser utilizados para manejar eventos en la interfaz de usuario, como el clic de un botón o la selección de un elemento en una lista desplegable.

## Inyección de dependencias JSF

La inyección de dependencias es un patrón de diseño que permite a los objetos obtener sus dependencias de forma transparente, en lugar de crearlas o buscarlas ellos mismos. Esto permite una mayor flexibilidad y modularidad en la aplicación, ya que las clases pueden ser desacopladas y reutilizadas más fácilmente.

En JSF, la inyección de dependencias se puede realizar utilizando un marco de inyección de dependencias, como CDI (Contexts and Dependency Injection). Los objetos que se pueden inyectar como dependencias se conocen como beans y pueden ser utilizados para proporcionar funcionalidad y lógica de negocio en la interfaz de usuario.

Por ejemplo, si tenemos un bean que se encarga de recuperar datos de una base de datos y queremos utilizar esos datos en una página JSF, podemos inyectar el bean en un controlador de página JSF y utilizarlo para recuperar y procesar los datos. De esta forma, podemos reutilizar el bean en diferentes páginas y controladores sin tener que duplicar código.

## beans.xml (CDI configuration File)

beans.xml es un archivo de configuración que se utiliza en el Contexts and Dependency Injection for the Java EE Platform (CDI), que es un marco de inyección de dependencias para aplicaciones Java EE. Este archivo se utiliza para definir las clases que deben ser consideradas como beans CDI y para especificar cómo deben ser inyectadas sus dependencias.

Un bean CDI es un objeto que puede ser inyectado en otras partes de una aplicación para proporcionar servicios. Los beans CDI se utilizan para implementar la lógica de negocio y los servicios de una aplicación y se pueden inyectar en otras clases a través de anotaciones.

En el archivo beans.xml, se pueden especificar distintos elementos para configurar los beans CDI, como el alcance de los beans, el modo de inyección, la resolución de dependencias y la configuración de interceptores. Por ejemplo, se puede utilizar el elemento <alternatives> para especificar clases que deben ser consideradas como alternativas a los beans predeterminados.

## Class Validator

En JavaServer Faces (JSF), un validador es un componente que se utiliza para validar los datos de entrada en un formulario antes de ser enviados al servidor. Los validadores se asocian a componentes de entrada en una página JSF y se ejecutan automáticamente cada vez que el usuario envía el formulario.

JSF proporciona varios validadores integrados que se pueden utilizar de forma fácil, como el validador de longitud, el validador de número, el validador de fecha, el validador de email, etc. También se pueden crear validadores personalizados que se adapten a las necesidades específicas de la aplicación.

Para utilizar un validador en una página JSF, primero se debe asociar el validador al componente de entrada correspondiente mediante el uso de la etiqueta f:validator.

Ejemplo de utilización de un validador de longitud:

Tabla 3. Ejemplo de validador de longitud.

<h:inputText id="name" value="#{bean.name}">

    <f:validateLength minimum="5" maximum="15" />

</h:inputText>

En este ejemplo se asocia al componente inputText un validador de longitud que asegura que el valor ingresado tiene al menos 5 caracteres y al menos 15 caracteres.

También es posible asociar validadores personalizados, mediante la implementación de una clase validator que implemente la interfaz javax.faces.validator.Validator, y luego se asocia en el componente utilizando f:validator con un atributo binding apuntando a la clase del validador.

Tabla 4. Ejemplo de validadores personalizados.

<h:inputText id="name" value="#{bean.name}">

    <f:validator binding="#{nameValidator}" />

</h:inputText>

Por otro lado, se debe de implementar la clase y definir su comportamiento en el método validate.

Tabla 5. Implementación de la clase Validator.

public class NameValidator implements Validator {

  public void validate(

    FacesContext context,

    UIComponent component,

    Object value

  ) throws ValidatorException {

    // Validaciones}

}

# PARTE PRÁCTICA

En general, los validadores son una herramienta importante para garantizar la precisión y la integridad de los datos en una aplicación JSF. Sin embargo, es importante recordar que, aunque los validadores son eficaces en la detección de errores, no garantizan la seguridad de la aplicación.

## CREACIÓN DEL PROYECTO

Para crear un nuevo proyecto se deberá abrir el IDE Apache Netbeans, seguido de eso se deberá seleccionar la opción File, New Project como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 2. Creación de un nuevo proyecto.

Seguido de eso se mostrará la ventana donde se deberá elegir el tipo de aplicación que se desea crear en este caso se deberá seleccionar Java Web, Web Application como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 3. Selección del tipo de proyecto a crear.

Una vez seleccionado el tipo de proyecto que se va a crear se deberá configurar los parámetros del proyecto como el nombre, la ubicación.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 4. Configuración de los parámetros del proyecto.

Para continuar se deberá seleccionar el servidor, la versión de JEE que se utilizará como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 5. Configuración del servidor y versión de JEE.

Finalmente, para comenzar con el desarrollo y codificación del proyecto se deberá seleccionar el Framework que se utilizará.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 6. Configuración de los Frameworks a utilizar.

## ESTRUCTURACIÓN DEL PROYECTO

El proyecto se va a desarrollar siguiente una arquitectura MODELO-VISTA-CONTROLADOR con la finalidad de hacer del proyecto entendible y escalable para lo cual se procede a la creación de los distintos paquetes.

Se deberá hacer clic derecho sobre el proyecto en la opción de new y Java Package.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 7. Creación de paquetes.

Seguido de eso se asigna un nombre al paquete, lo cual serán tres paquetes, ec.edu.monster.controlador, ec.edu.monster.modelo y el paquete vistas que en este caso se encuentran en la carpeta Web Pages.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 8. Creación de los paquetes Modelo y Controlador.

## CODIFICACIÓN DEL PROYECTO

### **CODIFICACIÓN DEL MODELO**

Para comenzar con la codificación de los archivos del paquete modelo se deberá agregar un nuevo archivo para ello se deberá hacer clic derecho sobre el paquete modelo, new y Other

Graphical user interface, text, application, chat or text message

Description automatically generated

Figura 9. Creación del archivo en el paquete modelo.

Se elige el tipo de archivo que se encuentra en la categoría JavaServer Faces el cual es JSF CDI Bean.

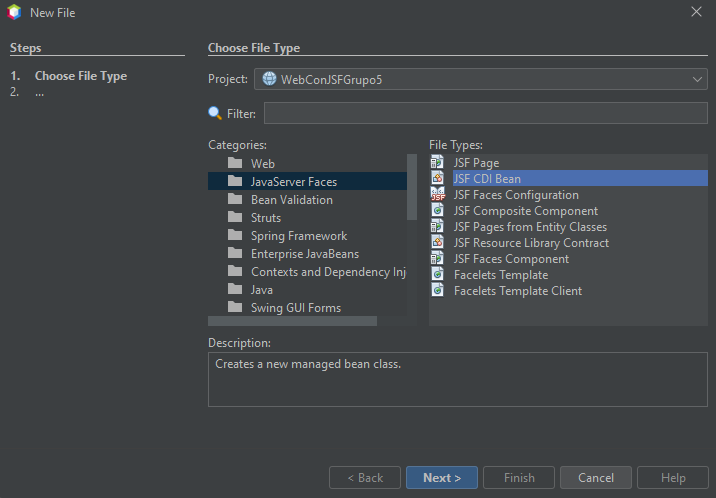


Figura 10. Selección del tipo de archivo.

Una vez seleccionado el tipo de archivo se deberá configurar los parámetros del archivo como el nombre, la ubicación entre otros.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 11. Configuración de parámetros del archivo.

Tabla 6. Codificación del archivo RegistrationBean.java

package ec.edu.monster.modelo;

import javax.enterprise.context.RequestScoped;

import javax.inject.Named;

/\*\*

 \*

 \* @author santi

 \*/

@Named(value = "registrationBean")

@RequestScoped

public class RegistrationBean {

  private String saludo;

  private String nombre;

  private String apellido;

  private Integer edad;

  private String email;

  public RegistrationBean() {}

  public String getSaludo() {

    return saludo;

  }

  public void setSaludo(String saludo) {

    this.saludo = saludo;

  }

  public String getNombre() {

    return nombre;

  }

  public void setNombre(String nombre) {

    this.nombre = nombre;

  }

  public String getApellido() {

    return apellido;

  }

  public void setApellido(String apellido) {

    this.apellido = apellido;

  }

  public Integer getEdad() {

    return edad;

  }

public void setEdad(Integer edad) {

    this.edad = edad;

  }

  public String getEmail() {

    return email;

  }

  public void setEmail(String email) {

    this.email = email;

  }

}

### **CODIFICACIÓN DE LA VISTA**

Para comenzar con la codificación de los archivos de la Vista se deberá crear los archivos haciendo clic derecho sobre la carpeta Web Pages, seguido de eso en New y Other.

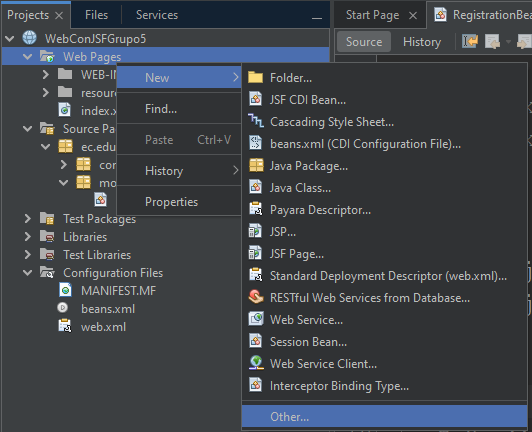


Figura 12. Creación de archivos para la Vista.

Seguido de eso se deberá seleccionar el tipo de archivo que se va a crear para ello se elige la categoría Web y HTML.

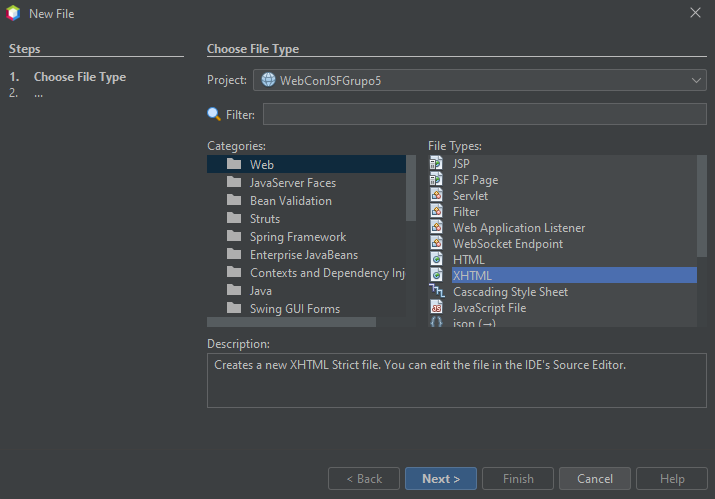


Figura 13. Selección del tipo de archivo que se va a crear.

Finalmente, se le asigna un nombre al archivo a crear.

Graphical user interface, text

Description automatically generated

Figura 14. Asignación de nombre al archivo.

Tabla 7. Codificación del archivo confirmación.html

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html>

<html

  xmlns="http://www.w3.org/1999/xhtml"

  xmlns:h="http://xmlns.jcp.org/jsf/html"

>

  <h:head>

    <title>Página de confirmación</title>

    <h:outputStylesheet library="css" name="estilos.css" />

  </h:head>

  <h:body>

    <h2>Página de confirmación</h2>

    <h:panelGrid columns="2" columnClasses=".alinder-bold,normal">

      <h:outputText value="Saludo: " />

      <h:outputText

        value="#{registrationBeanController.getRegistrationBean()

.saludo}"

      />

      <h:outputText value="Nombre: " />

      <h:outputText

        value="#{registrationBeanController.getRegistrationBean()

.nombre}"

      />

      <h:outputText value="Apellido: " />

      <h:outputText

        value="#{registrationBeanController.getRegistrationBean()

.apellido}"

      />

      <h:outputText value="Edad: " />

      <h:outputText

        value="#{registrationBeanController.getRegistrationBean()

.edad}"

      />

      <h:outputText value="Email: " />

      <h:outputText

        value="#{registrationBeanController.getRegistrationBean()

.email}"

      />

    </h:panelGrid>

  </h:body>

</html>

### **CODIFCACIÓN DEL CONTROLADOR**

Para comenzar con la codificación de los archivos del controlador se deberá hacer clic derecho sobre el paquete controlador, seguido de eso en new y elegir JSF CDI Bean.

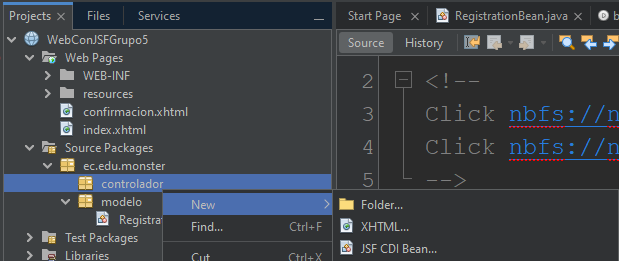


Figura 15. Creación del archivo para el paquete controlador.

Una vez seleccionado el tipo de archivo que se va a crear se procede a realizar la configuración de los parámetros del archivo, como el nombre, el paquete, la ubicación entre otros.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 16. Configuración de parámetros del archivo a crear.

Tabla 8. Codificación del archivo registrationBeanController.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.RegistrationBean;

import javax.enterprise.context.RequestScoped;

import javax.inject.Inject;

import javax.inject.Named;

@Named(value = "registrationBeanController")

@RequestScoped

public class registrationBeanController {

  @Inject

  private RegistrationBean registrationBean;

  public registrationBeanController() {}

  public RegistrationBean getRegistrationBean() {

    return registrationBean;

  }

  public void setRegistrationBean(RegistrationBean registrationBean) {

    this.registrationBean = registrationBean;

  }

  public String grabarRegistro() {

    return "confirmacion";

  }

}

## EJECUCIÓN DEL PROYECTO MANAGED BEANS

Para ejecutar el proyecto se deberá realizar los siguientes pasos:

1. Clic derecho sobre el proyecto, seguido de eso se deberá seleccionar la opción Clean and Build.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 17. Clean and Build del proyecto

1. Se deberá hacer clic sobre la opción de RUN y se mostrará la siguiente pantalla.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 18. Pantalla principal de la aplicación.

Una vez iniciado el proyecto se procede a llenar la información solicitada en el formulario.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 19. Llenar información solicitada.

Después se procede hacer clic en la opción de registrar y se mostrará la siguiente pantalla.

Graphical user interface, text

Description automatically generated

Figura 20. Ejecución de la aplicación.

Para continuar con la práctica se procede a realizar la validación del correo para lo cual se deberá crear un nuevo archivo en el controlador, se hace clic derecho sobre el paquete controlado, new y other.

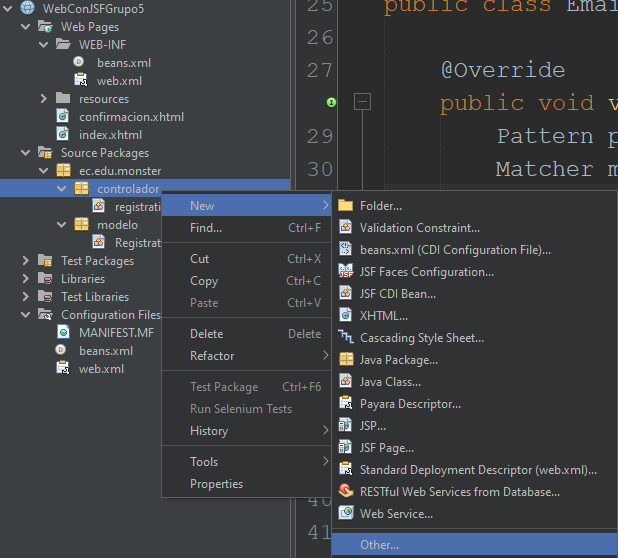


Figura 21. Creación de nuevo archivo para validar el correo.

Para validar el correo se hará uso de una clase de java para ello se deberá elegir la categoría Java y Java Class.

Graphical user interface, text, application

Description automatically generated

Figura 22. Selección del tipo de archivo.

Seguido de eso se deberá configurar los parámetros del archivo a crear como es el nombre, ubicación, entre otros.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 23. Configuración y asignación de nombre del archivo a crear.

Tabla 9. Codificación del archivo EmailValidator.java

package ec.edu.monster.controlador;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.component.html.HtmlInputText;

import javax.faces.context.FacesContext;

import javax.faces.validator.FacesValidator;

import javax.faces.validator.Validator;

import javax.faces.validator.ValidatorException;

/\*\*

 \*

 \* @author santi

 \*/

@FacesValidator("emailValidator")

public class EmailValidator implements Validator {

  @Override

  public void validate(

    FacesContext facesContext,

    UIComponent uIComponent,

    Object value

  ) throws ValidatorException {

    Pattern pattern = Pattern.compile("\\w+@\\w+\\.\\w+");

    Matcher matcher = pattern.matcher((CharSequence) value);

    HtmlInputText htmlInputText = (HtmlInputText) uIComponent;

    String label;

    if (

      htmlInputText.getLabel() == null ||

      htmlInputText.getLabel().trim().equals("")

    ) {

      label = htmlInputText.getId();

    } else {

      label = htmlInputText.getLabel();

    }

    if (!matcher.matches()) {

      FacesMessage facesMessage = new FacesMessage(

        label + ":no es una dirección email valida"

      );

      throw new ValidatorException(facesMessage);

    }

  }

}

Una vez implementado la validación de correo electrónico se procede a verificar su funcionamiento para ellos se deberá agregar el método creado al TAG.

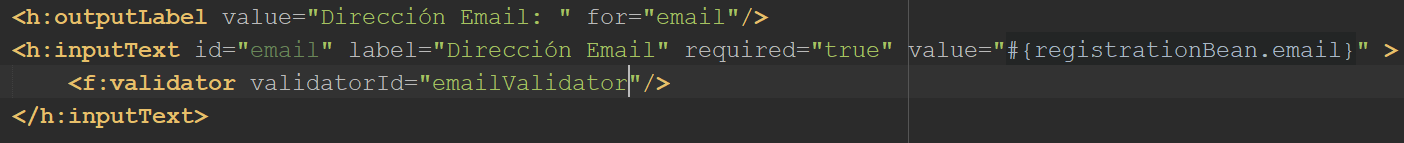


Figura 24. Agregar la validación al TAG.

Se procede a comprobar la validación del correo ingresando un correo incorrecto.

Graphical user interface, text, application, website

Description automatically generated

Figura 25. Comprobar validación de correo.

# CONCLUSIONES

* Se logró implementar un formulario web donde se aplican los conceptos de modelo, vista y controlador que es un patrón de diseño.
* Se logró realizar una consulta teórica de Java Server Faces, inyección de dependencias y las etiquetas que se manejan.
* Jakarta EE es una plataforma de tecnología empresarial basada en Java que se utiliza para el desarrollo de aplicaciones empresariales. Es una evolución de la plataforma Java Enterprise Edition (Java EE) y está diseñada para proporcionar un marco de trabajo completo y fácil de usar para el desarrollo de aplicaciones empresariales escalables y de alta calidad.
* El uso de Facelets template nos ayudan a mejorar el rendimiento ya que las Facelets se compilan en tiempo de ejecución lo cual permite tener una carga más rápida de las páginas.
* El hacer uso de Facelets ayuda en la mejora de la productividad ya que permite la separación entre el diseño y la lógica de la aplicación.

# RECOMENDACIONES

* Es recomendable hacer una revisión de los conceptos que se van a manejar dentro del desarrollo del proyecto.
* Es recomendable conocer las etiquetas básicas que se utiliza en Java Server Faces.
* Es recomendable conocer sobre el uso de dependencias.
* Es recomendable conocer sobre lenguajes de marcado como HTML, XML y del lenguaje de programación java.
* Es recomendable conocer acerca de la clase Validator ya que podemos implementar los métodos y personalizarlos para aplicar a nuestros proyectos.

# REFERENCIAS

* Pavón, Juan. (2009) Estructura de las Aplicaciones Orientadas a Objetos. El Patrón Modelo-Vista-Controlador (MVC). Universidad Complutense Madrid.
* Sommerville, Ian (2005). Ingeniera del Software. Pearson Educación.
* "Design Patterns: Elements of Reusable Object-Oriented Software" de Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides (también conocido como el "libro de los cuatro autores" o "libro Gang of Four").
* "Model-View-Controller (MVC) Explained" de James Tam.
* "Core JavaServer Faces" de David Geary y Cay S. Horstmann.
* "JavaServer Faces in Action" de Kito D. Mann.
* https://docs.oracle.com/javase/7/docs/api/javax/xml/validation/Validator.html