

**Tema**

LOGIN EN NETBEANS

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS.MDU.CCNA.CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

Fecha

27/07/2023

LOGIN EN NETBEANS.

[1. INTRODUCCIÓN 6](#_Toc127384495)

[2. OBJETIVO 6](#_Toc127384496)

[2.1 OBJETIVO GENERAL 6](#_Toc127384497)

[2.2 OBJETIVOS ESPECÍFICOS 6](#_Toc127384498)

[3. MARCO TEÓRICO 6](#_Toc127384499)

[3.1 ARQUITECTURA EN MODELO-VISTA-CONTROLADOR 6](#_Toc127384500)

[3.2 INTEGRACION A BASES DE DATOS DESDE JAVA. 9](#_Toc127384501)

[3.3 DATA ACCESS OBJECT. 9](#_Toc127384502)

[3.4 INTERFACES JAVA. 10](#_Toc127384503)

[3.5 JAVA SERVER FACES. 11](#_Toc127384504)

[3.6 FACELETS. 11](#_Toc127384505)

[3.7 JAVA PERSISTENCE API (JPA). 11](#_Toc127384506)

[3.8 OBJETOS DE MAPEO RELACIONAL. 12](#_Toc127384507)

[3.9 ENTITY MANAGER. 14](#_Toc127384508)

[3.10 MYSQL. 15](#_Toc127384509)

[3.11 DLL Y DML 16](#_Toc127384510)

[3.12 IMPORTANCIA DE LOS ROLES EN UNA APLICACIÓN. 16](#_Toc127384511)

[3.13 MYSQL CONNECTOR Y JDBC. 17](#_Toc127384512)

[3.14 ROLES Y USUARIOS RELACIÓN BASE GENERAL. 18](#_Toc127384513)

[3.15 PROGRAMACION ORIENTADA A OBJETOS (POO). 19](#_Toc127384514)

[3.10 PROTOCOLO SIMPLE DE TRANSFERENCIA DE CORREO (SMTP). 22](#_Toc127384515)

[3.11 PROTOCOLO POST OFFICE (POP). 24](#_Toc127384516)

[3.12 IMAP. 25](#_Toc127384517)

[3.13 CIFRADO SSL Y TLS. 26](#_Toc127384518)

[3.14 MÉTODOS. 27](#_Toc127384519)

[3.15 TÉRMINOS. 28](#_Toc127384520)

[4. DESARROLLO 29](#_Toc127384521)

[4.1 CREACIÓN DEL PROYECTO 29](#_Toc127384522)

[4.1.1 CREAR UNA APLICACIÓN LLAMADA LOGIN\_NETBEANS\_GRUPO#. 29](#_Toc127384523)

[4.1.2 CREACIÓN DE ESTRUCTURA MVC DEL PROYECTO. 31](#_Toc127384524)

[4.1.3 IMPORTACIÓN DE BIBLIOTECAS NECESARIAS. 32](#_Toc127384525)

[4.1.4 CONEXIÓN CON LA BASE DE DATOS. 33](#_Toc127384526)

[4.2 CREACIÓN DE VISTAS DEL PROYECTO 35](#_Toc127384527)

[4.2.1 CREACIÓN DE ARCHIVOS JSF. 35](#_Toc127384528)

[4.2.2 CODIFICACIÓN DE ARCHIVO INDEX 39](#_Toc127384529)

[4.2.3 CREACIÓN DE ARCHIVO MENU. 42](#_Toc127384530)

[4.2.4 CREACIÓN DE ARCHIVO CAMBIARPASSWORD. 45](#_Toc127384531)

[4.2.5 CREACIÓN DE ARCHIVO RESETEARPASSWORD. 47](#_Toc127384532)

[4.2.6 MODIFICACIÓN DE CRUDS. 50](#_Toc127384533)

[4.2.7 IMPLEMENTACIÓN DE TABVIEWS. 53](#_Toc127384534)

[4.3 CREACIÓN DE MODELOS DEL PROYECTO 66](#_Toc127384535)

[4.3.1 CREACIÓN DE LOS ARCHIVOS DEL MODELO. 66](#_Toc127384536)

[4.3.2 CREACIÓN DEL ARCHIVO CORREO 70](#_Toc127384537)

[4.4 CREACIÓN DE CONTROLADORES DEL PROYECTO 75](#_Toc127384538)

[4.4.1 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO MENUCONTROLADOR. 75](#_Toc127384539)

[4.4.2 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO ENVIARCORREO. 82](#_Toc127384540)

[4.4.3 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO EMAILVALIDATOR. 86](#_Toc127384541)

[4.4.4 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO CEDULAVALIDATOR. 89](#_Toc127384542)

[4.4.5 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO PASSWORDVALIDATOR. 93](#_Toc127384543)

[5. EJECUCIÓN DEL PROYECTO 97](#_Toc127384544)

[6. CONCLUSIONES 101](#_Toc127384545)

[7. RECOMENDACIONES 101](#_Toc127384546)

[8. BIBLIOGRAFÍA 101](#_Toc127384547)

**ÍNDICE DE IMÁGENES**

[Figura 1. Estructura de MVC 7](#_Toc127384422)

[Figura 2. Código sencillo sobre el funcionamiento de timers. 10](#_Toc127384423)

[Figura 3. Estructura JPA 12](#_Toc127384424)

[Figura 4. Interacción del ORM 13](#_Toc127384425)

[Figura 5. Estructura EntityManager 14](#_Toc127384426)

[Figura 6. Diagrama básico de sockets. 18](#_Toc127384427)

[Figura 7. Código sencillo sobre el funcionamiento de timers. 19](#_Toc127384428)

[Figura 8. Características de POO. 22](#_Toc127384429)

[Figura 9. Funcionamiento Correo electrónico. 23](#_Toc127384430)

[Figura 10. Logotipo identificativo de Apache Netbeans 30](#_Toc127384431)

[Figura 11. Selección del tipo de aplicación a desarrollar 30](#_Toc127384432)

[Figura 12. Selección del nombre y directorio donde guardar el proyecto. 31](#_Toc127384433)

[Figura 13. Estructura del proyecto generado. 31](#_Toc127384434)

[Figura 14. Creación de carpeta 32](#_Toc127384435)

[Figura 15. Estructura del proyecto. 32](#_Toc127384436)

[Figura 16. Creación de carpeta 33](#_Toc127384437)

[Figura 17. Importación de librerías 33](#_Toc127384438)

[Figura 18. Creación de una nueva conexión 33](#_Toc127384439)

[Figura 19. Driver de conexión 34](#_Toc127384440)

[Figura 20. Configuración de conexión 34](#_Toc127384441)

[Figura 21. Conexión con la BD 35](#_Toc127384442)

[Figura 22. Creación de Vistas. 36](#_Toc127384443)

[Figura 23. Selección de tipo de archivo 36](#_Toc127384444)

[Figura 24. Selección de clases. 37](#_Toc127384445)

[Figura 25. Configuración de paquetes JSF 37](#_Toc127384446)

[Figura 26. Esquema del proyecto con las vistas creada 38](#_Toc127384447)

[Figura 27. Codificación del Login en index.xhtml. 39](#_Toc127384448)

[*Figura 28. Cambio de nombre del archivo.* 42](#_Toc127384449)

[*Figura 29. Creación archivo menú.xhtml* 42](#_Toc127384450)

[*Figura 30. Selección nuevo JSP Pages.* 45](#_Toc127384451)

[*Figura 31. Creación archivo cambiarPassword.xhtml* 45](#_Toc127384452)

[*Figura 32. Selección nueva JSP Pages.* 47](#_Toc127384453)

[*Figura 33. Creación archivo resetarPassword.xhtml* 47](#_Toc127384454)

[*Figura 34. CRUD original.* 50](#_Toc127384455)

[*Figura 35. CRUD modificado* 50](#_Toc127384456)

[*Figura 36. Ubicación del archivo.* 53](#_Toc127384457)

[*Figura 37. Implementación de los TabView* 53](#_Toc127384458)

[*Figura 38. Vista de los formularios con TabView* 54](#_Toc127384459)

[Figura 39. Selección de un archivo para clase. 66](#_Toc127384460)

[Figura 40. Selección de tipo de archivo. 66](#_Toc127384461)

[Figura 41. Elección de base de datos. 67](#_Toc127384462)

[Figura 42. Configuración de base de datos y conexión. 67](#_Toc127384463)

[Figura 43. Selección de tablas. 68](#_Toc127384464)

[Figura 44. Ubicación de archivos. 68](#_Toc127384465)

[Figura 45. Esquema del paquete ec.edu.monster.modelo . 69](#_Toc127384466)

[Figura 46. Selección de un archivo para clase. 70](#_Toc127384467)

[Figura 47. Nombre y ubicación de archivo. 70](#_Toc127384468)

[Figura 48. Codificación del archivo Correo.java. 71](#_Toc127384469)

[Figura 49. Selección de un archivo Java Class. 75](#_Toc127384470)

[Figura 50. Creación del archivo MenuControlador,java 76](#_Toc127384471)

[Figura 51. Codificación del archivo MenuControlador,java 76](#_Toc127384472)

[Figura 52. Selección de un archivo Java Class. 82](#_Toc127384473)

[Figura 53. Creación del archivo EnviarCorreo,java 82](#_Toc127384474)

[Figura 54. Codificación del archivo EnviarCorreo,java 83](#_Toc127384475)

[Figura 55. Selección de un archivo Java Class. 86](#_Toc127384476)

[Figura 56. Creación del archivo EmailValidator,java 86](#_Toc127384477)

[Figura 57. Codificación del archivo EmailValidator,java 87](#_Toc127384478)

[Figura 58. Selección de un archivo Java Class. 89](#_Toc127384479)

[Figura 59. Creación del archivo CedulaValidator,java 89](#_Toc127384480)

[Figura 60. Codificación del archivo CedulaValidator,java 90](#_Toc127384481)

[Figura 61. Selección de un archivo Java Class. 93](#_Toc127384482)

[Figura 62. Creación del archivo PasswordValidator,java 93](#_Toc127384483)

[Figura 63. Codificación del archivo PasswordValidator,java 94](#_Toc127384484)

[Figura 64. Botón para ejecutar el proyecto. 97](#_Toc127384485)

[Figura 65. Ejecución del proyecto. 97](#_Toc127384486)

[Figura 66. Mensajes de Advertencia. 98](#_Toc127384487)

[Figura 67. Menú de Administrador. 98](#_Toc127384488)

[Figura 68. Menú de Empleado. 99](#_Toc127384489)

[Figura 69. Registro de usuario. 99](#_Toc127384490)

[Figura 70. Crud de usuario. 100](#_Toc127384491)

[Figura 71. Marco de pantalla de información de usuario. 100](#_Toc127384492)

[Figura 72. Cambio de contraseña. 100](#_Toc127384493)

[Figura 73. Correo de reseteo de contraseña. 101](#_Toc127384494)

**ÍNDICE DE TABLAS**

[Tabla 1. Mandatos de SMTP 23](#_Toc127384249)

[Tabla 2. Métodos generalmente utilizados en conexión de java y bases de datos. 27](#_Toc127384250)

[Tabla 3. Términos 28](#_Toc127384251)

[Tabla 4. Codificación del archivo index 40](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384252)

[Tabla 5. Codificación del archivo menú.xhtml 43](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384253)

[Tabla 6. Codificación del archivo cambiarPassword.xhtml 46](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384254)

[Tabla 7. Codificación del archivo resetarPassword.xhtml 48](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384255)

[Tabla 8. Codificación del archivo peempEmple/List.xhtml 51](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384256)

[Tabla 9. Codificación del archivo peempEmple/ View.xhtml 55](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384257)

[Tabla 10. Codificación del archivo peempEmple/ Create.xhtml 58](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384258)

[Tabla 11. Codificación del archivo peempEmple/ Edit.xhtml 62](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384259)

[Tabla 12. Codificación del archivo Correo.java 72](#_Toc127384260)

[Tabla 13. Codificación del archivo controlador MenuControlador.java 77](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384261)

[Tabla . Codificación del archivo controlador EnviarCorreo.java 84](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384262)

[Tabla 15. Codificación del archivo controlador EmailValidator.java 88](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384263)

[Tabla 16. Codificación del archivo controlador CedulaValidator.java 91](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384264)

[Tabla 17. Codificación del archivo controlador PasswordValidator.java 95](https://uespe-my.sharepoint.com/personal/cpnovoa_espe_edu_ec/Documents/LOGIN%20EN%20NETBEANS_GRUPO6.docx#_Toc127384265)

# INTRODUCCIÓN

En este trabajo de investigación y desarrollo, se presenta la creación de una aplicación web desarrollada en Java EE para el sistema de inicio de sesión de la empresa Screams Inc. La aplicación cuenta con una conexión segura a una base de datos MySQL para el almacenamiento y recuperación de información de los usuarios. El trabajo se estructura en tres partes: en primer lugar, se presenta un marco teórico que aborda los componentes y herramientas utilizadas en el desarrollo de la aplicación. En segundo lugar, se proporciona una guía detallada sobre los pasos seguidos en el proceso de desarrollo de la aplicación. Por último, se presentan las conclusiones obtenidas a partir del trabajo realizado. El objetivo principal de esta aplicación es proporcionar un mecanismo de inicio de sesión fácil y seguro para los empleados de la empresa, así como garantizar la protección de la información de los usuarios. A través de este trabajo se pretende mejorar la eficiencia y seguridad en la gestión de usuarios en Screams Inc.

# OBJETIVO

### OBJETIVO GENERAL

Diseñar e implementar un sistema de inicio de sesión seguro y eficiente para la empresa Screams Inc utilizando Java EE y una conexión a una base de datos MySQL.

### OBJETIVOS ESPECÍFICOS

* Investigar los componentes y herramientas necesarios para desarrollar una aplicación web de inicio de sesión en Java EE.
* Desarrollar una aplicación web que permita a los usuarios iniciar sesión de manera segura y eficiente en la plataforma de Screams Inc.
* Proporcionar una guía detallada sobre el proceso de desarrollo de la aplicación.
* Proporcionar conclusiones y recomendaciones para mejorar el sistema en el futuro.

# MARCO TEÓRICO

## 3.1 ARQUITECTURA EN MODELO-VISTA-CONTROLADOR

El patrón de arquitectura MVC [1] es un patrón de diseño de software que se utiliza comúnmente para desarrollar aplicaciones web y de escritorio. La idea detrás de MVC es dividir la aplicación en tres componentes principales, cada uno con una función específica. El patrón de arquitectura MVC convierte el desarrollo de aplicaciones complejas en un proceso mucho más manejable. Permite a varios desarrolladores trabajar simultáneamente en la aplicación.

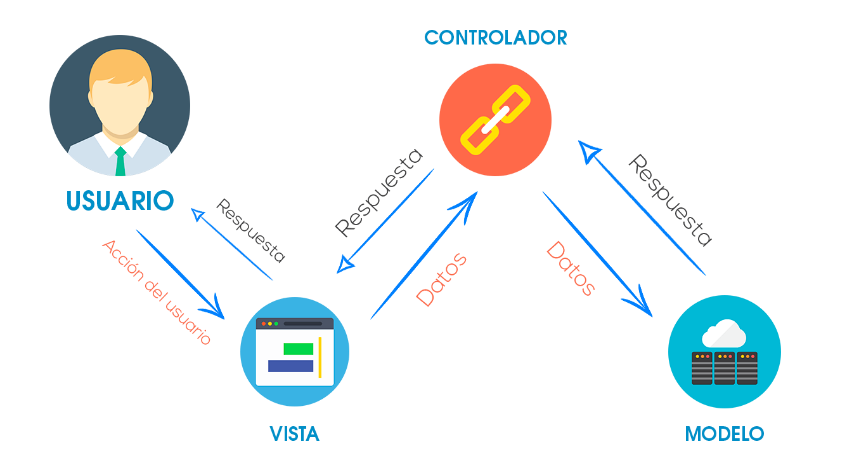


Figura 1. Estructura de MVC

Componentes:

* **Modelo**: es la parte de la aplicación que se encarga de gestionar los datos y la lógica de negocio. Es responsable de la lectura, escritura y actualización de los datos, así como de la realización de cálculos y validaciones necesarias para mantener la integridad de los datos. Puede realizar operaciones complejas en los datos, como la creación de informes, el análisis de datos y la generación de estadísticas. La función principal del modelo es proporcionar una interfaz para que los demás componentes de la aplicación puedan interactuar con los datos y se comunica con la vista y el controlador a través de interfaces definidas, como APIs o protocolos de red. Por ejemplo, cuando un usuario envía una solicitud al controlador, el controlador interactúa con el modelo para obtener los datos necesarios y luego los devuelve a la vista para que se muestren al usuario.

Un ejemplo de uso del modelo en la arquitectura MVC podría ser una aplicación de carrito de compras en línea. En este caso, el modelo sería responsable de gestionar la información de los productos, como su nombre, descripción, precio y disponibilidad. El modelo también podría realizar cálculos para determinar el total de la compra y aplicar descuentos y promociones. La vista mostraría los productos disponibles y permitiría al usuario seleccionar los productos que desea comprar. El controlador procesaría la selección del usuario y comunicaría con el modelo para actualizar la información del carrito de compras y realizar el cálculo del total. Finalmente, la vista mostraría al usuario el resumen de su compra y permitiría que se realice la transacción.

* **Vista**: Se encarga de presentar los datos y la interfaz de usuario al usuario final. La vista es responsable de mostrar la información de manera clara y comprensible para el usuario, y de permitir al usuario interactuar con la aplicación de una manera intuitiva y eficiente. No realiza operaciones de lógica de negocio o manejo de datos, sino que se limita a mostrar los datos y a recibir entradas del usuario. Por lo tanto, la vista no tiene conocimiento directo del modelo, sino que se comunica con el modelo a través del controlador. Puede ser implementada de muchas maneras, desde una simple página HTML hasta una aplicación de escritorio o móvil. La vista puede estar diseñada para ser atractiva visualmente y fácil de usar, y puede incluir elementos interactivos como botones, formularios y menús desplegables.

Un ejemplo de uso de la vista en la arquitectura MVC podría ser una aplicación de redes sociales. En este caso, la vista sería responsable de mostrar la información de la cuenta del usuario, como su nombre, foto de perfil, amigos y publicaciones. La vista también permitiría al usuario publicar nuevas entradas, comentar las publicaciones de otros usuarios y enviar mensajes a sus amigos. El controlador procesaría estas entradas del usuario y comunicaría con el modelo para actualizar los datos del usuario. La vista, a su vez, mostraría los resultados actualizados al usuario.

* **Controlador**: Es el componente que se encarga de recibir y procesar las solicitudes del usuario, y de comunicarse con el modelo y la vista según sea necesario. El controlador es responsable de actuar como intermediario entre la vista y el modelo. Recibe las solicitudes del usuario y realiza una o varias operaciones para satisfacer la solicitud. Estas operaciones pueden incluir la lectura o escritura de datos en el modelo, o la comunicación con la vista para mostrar información al usuario. El controlador también puede realizar validaciones y otras operaciones de lógica de negocio para garantizar la integridad de los datos. El controlador se comunica con el modelo y la vista a través de interfaces definidas, como APIs o protocolos de red. Por ejemplo, cuando un usuario envía una solicitud a través de la vista, el controlador procesa la solicitud y se comunica con el modelo para obtener o actualizar los datos necesarios. Luego, el controlador devuelve los resultados a la vista para que sean presentados al usuario.

Un ejemplo de uso del controlador en la arquitectura MVC podría ser una aplicación de registro de usuarios en línea. En este caso, el controlador sería responsable de recibir los datos de registro del usuario, como su nombre, dirección de correo electrónico y contraseña. El controlador realizaría validaciones en los datos para garantizar que sean correctos y completos, y luego comunicaría con el modelo para almacenar los datos en la base de datos. Si la operación es exitosa, el controlador redirigiría al usuario a una página de confirmación. Si la operación falla, el controlador volvería a mostrar el formulario de registro con un mensaje de error.

Características de la arquitectura MVC:

* **Separación de preocupaciones:** Cada componente tiene una función específica, lo que facilita el mantenimiento y la escalabilidad del código.
* **Modularidad:** Cada componente puede ser desarrollado de forma independiente, lo que permite a los desarrolladores trabajar en diferentes partes de la aplicación sin interferir en el trabajo de los demás.
* **Reutilización:** Los componentes pueden ser reutilizados en diferentes partes de la aplicación o en diferentes aplicaciones, lo que ahorra tiempo y esfuerzo de desarrollo.

Ventajas de usar MVC:

* Facilita la colaboración entre desarrolladores, ya que cada uno puede trabajar en diferentes partes de la aplicación sin interferir en el trabajo de los demás.
* Facilita el mantenimiento y la escalabilidad del código, ya que cada componente tiene una función específica y se puede modificar o actualizar sin afectar a los demás componentes.
* Facilita las pruebas, ya que cada componente puede ser probado de forma independiente.

Diferencias con otras Arquitecturas:

* **MVC vs. Arquitectura de tres capas**: Ambas arquitecturas dividen la aplicación en tres componentes, pero la arquitectura de tres capas tiene una capa adicional, la capa de acceso a datos, que es responsable de la conexión con la base de datos y la recuperación de datos. En MVC, esta función se realiza en el modelo.
* **MVC vs. Arquitectura cliente-servidor**: En la arquitectura cliente-servidor, la lógica de negocio y los datos residen en un servidor y la interfaz de usuario se ejecuta en un cliente (por ejemplo, un navegador web). En MVC, la lógica de negocio y los datos residen en el modelo, pero la interfaz de usuario se ejecuta en la vista en el lado del cliente.

**MVC vs. Arquitectura basada en servicios**: En una arquitectura basada en servicios, los componentes de la aplicación se comunican a través de servicios web o API. En MVC, los componentes se comunican directamente entre sí.

## 3.2 INTEGRACION A BASES DE DATOS DESDE JAVA.

Cuando se habla de la conexión de una aplicación con una base de datos no [2] existe una única forma de realizar este proceso, ya que existen varias formas de conectar una aplicación web desarrollada en Java con una base de datos, algunos ejemplos son:

* JDBC (Java Database Connectivity): Es una interfaz de programación de aplicaciones (API) que permite a los desarrolladores de Java interactuar con cualquier base de datos soportada mediante un controlador JDBC. Con JDBC, una aplicación puede enviar comandos SQL a una base de datos y procesar los resultados.
* JPA (Java Persistence API): Es una interfaz de programación de aplicaciones que proporciona una manera de acceder a una base de datos mediante objetos Java. JPA utiliza un modelo de objeto-relacional (ORM) para mapear las tablas de una base de datos con las clases Java correspondientes.
* Hibernate: Es una implementación popular de JPA. Hibernate proporciona una interfaz para interactuar con una base de datos mediante objetos Java y mapea automáticamente los objetos a las tablas de la base de datos.
* Spring Data: Es un marco de trabajo que proporciona una interfaz para interactuar con una base de datos mediante objetos Java. Spring Data se basa en JPA y Hibernate, y proporciona una interfaz de programación de aplicaciones consistente para diferentes tipos de bases de datos.

En general, todas estas herramientas proveen una capa de abstracción sobre JDBC y permiten a los desarrolladores interactuar con la base de datos mediante objetos Java, lo que facilita la escritura de código y reduce los errores. Es importante que se escoja la herramienta adecuada para el proyecto y se tenga en cuenta las necesidades y requerimientos del proyecto.

## DATA ACCESS OBJECT.

DAO (Data Access Object) es un [3] patrón de diseño en Java que se utiliza para separar la lógica de acceso a datos de la lógica de negocio de una aplicación. El objetivo de este patrón es proporcionar una interfaz para acceder a los datos almacenados en una base de datos de manera independiente de la implementación de la base de datos.

Un DAO es una clase Java que se encarga de realizar operaciones CRUD (create, read, update, delete) en una base de datos. Esta clase proporciona métodos para insertar, actualizar, eliminar y recuperar datos de la base de datos. Un DAO suele tener una conexión a la base de datos y utiliza JDBC para enviar y recibir datos.

El DAO ayuda a aislar la lógica de negocio de la aplicación de los detalles de acceso a datos y permite que la lógica de negocio se pueda reutilizar fácilmente con diferentes bases de datos. Por ejemplo, si se decide cambiar de una base de datos MySQL a una base de datos Oracle, solo se necesitaría cambiar la implementación del DAO para adaptarse a la nueva base de datos, en lugar de tener que cambiar toda la aplicación.

En resumen, el DAO es una clase que se encarga de abstraer los detalles de acceso a datos y se utiliza para separar la lógica de negocio de la aplicación de la lógica de acceso a datos, permitiendo una mayor flexibilidad y escalabilidad en el desarrollo de aplicaciones..

## INTERFACES JAVA.

Las interfaces en Java se [4] utilizan para definir un contrato de métodos que una clase debe cumplir. Por ejemplo, si varias clases deben tener un método "start" que se ejecuta de manera diferente en cada clase, se puede crear una interfaz "Startable" con un método "start" y hacer que cada clase implemente esa interfaz. De esta manera, se garantiza que todas las clases tengan un método "start" y se pueden utilizar de manera polimórfica.

Una interfaz se define utilizando la palabra clave "interface" seguida del nombre de la interfaz y los métodos que esta incluye. Los métodos en una interfaz solo tienen la declaración y no tienen una implementación.

Para utilizar una interfaz en una clase, se utiliza la palabra clave "implements" seguida del nombre de la interfaz. Luego, la clase debe proporcionar una implementación para cada uno de los métodos de la interfaz.

Un ejemplo de su uso en el mundo real es en la programación de aplicaciones móviles. Por ejemplo, se podría crear una interfaz "Connectable" que incluya métodos para conectar y desconectar un dispositivo. Luego, se podrían crear clases específicas para cada tipo de dispositivo (como Bluetooth, WiFi, etc.) que implementen esa interfaz.

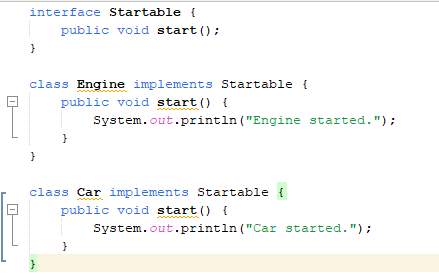


Figura 2. Código sencillo sobre el funcionamiento de timers.

En este ejemplo, se define una interfaz "Startable" con un método "start". Luego, se crean las clases "Engine" y "Car" que implementan esa interfaz y proporcionan una implementación específica para el método "start".

## JAVA SERVER FACES.

Java Server Faces o por abreviatura JSF es un framework utilizado en Java enfocado al desarrollo de UI. Al contrario de frameworks de desarrollo más actuales cuyo enfoque es que se ejecutan del lado del cliente, JSF se ejecuta desde el lado de servidor lo cual le da una estructura más organizada y mantenible puesto que permite tener una separación entre el comportamiento de la aplicación y su forma de ser presentada.

Tal y como [5] indica contiene una serie de características que la diferencian de otros frameworks de desarrollo de UI entre las que se puede destacar:

* Una API para: Representar componentes de Interfaz de Usuario (UI) y gestionar su estado.
* Manejar eventos, validar en el servidor y conversión de datos.
* Definir la navegación de páginas.
* Soporte de internacionalización y accesibilidad.
* Dos librerías de etiquetas JSP personalizadas para expresar componentes en una página JSP y enlazar los componentes a objetos del servidor.

Lo cual da como resultado que el modelo de programación se encuentre bien definido y las librerías de etiquetas faciliten la construcción y mantenimiento de las aplicaciones web con Interfaces de Usuario (UI) del servidor.

## FACELETS.

Facelets es una tecnología de renderizado de vistas para JavaServer Faces (JSF) que se utiliza para crear interfaces de usuario web. Es una alternativa a la tecnología de renderizado de vistas JSP (JavaServer Pages) que se utiliza para crear aplicaciones web con Java.

Algunas de las características que lo diferencian son:

* Utiliza un sistema de plantillas para crear una estructura de página común.
* Permite crear componentes reutilizables y personalizados.
* Ofrece una sintaxis de marcas más simple y fácil de aprender que JSP.
* Mejora el rendimiento al comparar con JSP ya que los archivos xhtml no son procesados por un intérprete adicional.

Para utilizar Facelets, se requiere una implementación de JSF en el servidor de aplicaciones. Una vez configurado, los desarrolladores pueden crear vistas xhtml y utilizar componentes JSF especiales para interactuar con el código Java en el servidor.

Además, Facelets es compatible con una gran cantidad de herramientas de desarrollo de terceros y frameworks como PrimeFaces, RichFaces, BootsFaces, etc.

Es una alternativa más moderna y eficiente a JSP y ofrece una sintaxis de marcas más simple y fácil de aprender y características adicionales como plantillas, componentes reutilizables y mejoras en rendimiento.

## JAVA PERSISTENCE API (JPA).

JPA es una API Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB) y sustituye a las, es decir automatiza la asignación de objetos java a tabla de objetos de base de datos relacionales [6].

Las características que forman parte del trabajo del la JPA son:

* **Metadatos:** JPA Asigna objetos a una base de datos mediante el uso de metadatos
* **Entidades y metadatos:**
  + Entidades tienen asociados metadatos que describen el mapeado
  + Permiten reconocer las asignaciones
* **Formatos:**
  + Anotaciones descritas en javax.persistence.
  + Descriptores XML
* **Configuración por excepción**:
  + JPA cuenta con reglas del mapeado por defecto
  + Solo se necesita metadatos para excepciones.

Diagrama

Descripción generada automáticamente con confianza baja

Figura . Estructura JPA

Cuando se empieza a trabajar con bases de datos en Java lo primero que nos enseñan es a utiliza el API de JDBC el cual nos permite realizar consultas directas a la base de datos a través de consultas SQL nativas. JDBC por mucho tiempo fue la única forma de interactuar con las bases de datos, pero representaba un gran problema y es que Java es un lenguaje orientado a objetos y se tenía que convertir los atributos de las clases en una consulta SQL como SELECT, INSERT, UPDATE, DELTE, etc. lo que ocasionaba un gran esfuerzo de trabajo y un provocaba muchos errores en tiempo de ejecución, debido principalmente a que las consultas SQL se tenían que generar frecuentemente al vuelo. Los componentes que forman parte de las JPA son ORM, EntityManager.

## OBJETOS DE MAPEO RELACIONAL.

Ser encarga de asignar objetos a los datos almacenados en base de datos relacionales, es decir automatizar objetos java a tablas de base de datos relacionales [2], estas a su ves sustituyen a las entity beans.

El ORM se encarga, de forma automática, de convertir los objetos en registros y viceversa, simulando así tener una base de datos orientada a objetos.

A la hora de almacenar los datos de un programa orientado a objetos en una base de datos relacional, surge un inconveniente debido a incompatibilidad de sistemas de tipos de datos. En el software orientado a objetos, la información se representa como clases y objetos. En las bases de datos relacionales, como tablas y sus restricciones. Por tanto, para almacenar la información tratada en un programa orientado a objetos en una base de datos relacional es necesaria una traducción entre ambas formas.

Diagrama

Descripción generada automáticamente

Figura . Interacción del ORM

El mapeo objeto-relacional (ORM) soluciona este problema. Es una técnica de programación que se utiliza con el propósito de convertir datos entre el utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, gracias a la la persistencia. Esto posibilita el uso en las bases de datos relacionales de las características propias de la programación orientada a objetos (básicamente herencia y polimorfismo).

Object-Relational mapping, o lo que es lo mismo, mapeo de objeto-relacional, es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Las bases de datos más extendidas son del tipo relacional y estas sólo permiten guardar tipos de datos primitivos (enteros, cadenas de texto, etc.) por lo que no se puede guardar de forma directa los objetos de la aplicación en las tablas. Por tanto, se debe convertir los valores de los objetos en valores simples que puedan ser almacenados en una base de datos (y poder recuperarlos más tarde).

El mapeo objeto-relacional surge, pues, para dar respuesta a esta problemática: traducir los objetos a formas que puedan ser almacenadas en bases de datos preservando las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes. Aunque el lenguaje SQL se usa para acceder a muchas de las bases de datos existentes, existen múltiples varianzas en las funciones que los distintos SGBD han usado.

## ENTITY MANAGER.

El funcionamiento del entity manager está especificado en una única interfaz llamada EntityManager (enlace a javadoc) [7].

El entity manager tiene dos responsabilidades fundamentales:

* Define una conexión transaccional con la base de datos que debe abrir y mantener abierta mientras se está realizado operaciones. En este sentido realiza funciones similares a las de una conexión JDBC.
* Además, mantiene en memoria una caché con las entidades que gestiona y es responsable de sincronizarlas correctamente con la base de datos cuando se realiza un flush. El conjunto de entidades que gestiona un entity manager se denomina su contexto de persistencia.

El entity manager se obtiene a través de una factoría del tipo EntityManagerFactory, que se configura mediante la especificación de una unidad de persistencia (persistence unit en inglés) definida en el fichero XML persistence.xml. En el fichero pueden haber definidas más de una unidad de persistencia, cada una con un nombre distinto. El nombre de la unidad de persistencia escogida se pasa a la factoría. La unidad de persistencia define las características concretas de la base de datos con la que van a trabajar todos los entity managers obtenidos a partir de esa factoría y queda asociada a ella en el momento de su creación. Existe, por tanto, una relación uno-a-uno entre una unidad de persistencia y su EntityManagerFactory concreto. Para obtener una factoría EntityManagerFactory se debe llamar a un método estático de la clase Persistence.

Diagrama

Descripción generada automáticamente

Figura . Estructura EntityManager

una vez creado el entity manager se utiliza para realizar todas las operaciones de recuperación, consulta y actualización de entidades. Cuando un entity manager obtiene una referencia a una entidad, se dice que la entidad está gestionada (una managed entity en inglés) por él. El entity manager guarda internamente todas las entidades que gestiona y las utiliza como una caché de los datos en la base de datos. Por ejemplo, cuando va a recuperar una entidad por su clave primaria, lo primero que hace es consultar en su caché si esta entidad ya la ha recuperado previamente. Si es así, no necesita hacer la búsqueda en la base de datos y devuelve la propia referencia que mantiene. Al conjunto de entidades gestionadas por un entity manager se le denomina su contexto de persistencia (persistence context en inglés).

En un determinado momento, el entity manager debe volcar a la base de datos todos los cambios que se han realizado sobre las entidades. También debe ejecutar las consultas JPQL definidas. Para ello el entity manager utiliza un proveedor de persistencia (persistence provider en inglés) que es el responsable de generar todo el código SQL compatible con la base de datos.

Contiene la colección de entidades gestionadas por el entity manager que están conectadas y sincronizadas con la base de datos. Cuando el entity manager cierra una transacción, su contexto de persistencia se sincroniza automáticamente con la base de datos. Sin embargo, a pesar del importante papel que juega, el contexto de persistencia nunca es realmente visible a la aplicación. Siempre se accede a él indirectamente a través del entity manager y se asume que está ahí cuando se necesita.

Es también fundamental entender que el contexto de persistencia hace el papel de caché de las entidades que están realmente en la base de datos. Cuando se actualiza una instancia en el contexto de persistencia se esta actualizando una caché, una copia que sólo se hace persistente en la base de datos cuando el entity manager realiza un flush de las instancias en la base de datos.

Cada vez que el proveedor de persistencia genera sentencias SQL y las escribe en la base de datos a través de una conexión JDBC, se dice que se ha volcado (flush) el contexto de persistencia. Todos los cambios pendientes que requieren que se ejecute una sentencia SQL en la transacción se escriben en la base de datos cuando ésta realiza un commit. Esto significa que cualquier operación SQL que tenga lugar después de haberse realizado el volcado ya incorporará estos cambios. Esto es particularmente importante para consultas SQL que se ejecutan en una transacción que también está realizando cambios en los datos de la entidad.

## MYSQL.

MySQL es [8] un sistema de gestión de bases de datos relacionales (RDBMS) de código abierto. Fue desarrollado por la empresa sueca MySQL AB, y actualmente es propiedad de Oracle. MySQL es uno de los sistemas de gestión de bases de datos más populares y ampliamente utilizados en el mundo, especialmente en aplicaciones web y sistemas empresariales.

Este gestor almacena datos en tablas, que a su vez están compuestas por columnas y filas. Cada tabla tiene una estructura definida mediante un esquema, que especifica los nombres de las columnas, el tipo de datos que almacenan, y otras características. MySQL también permite la creación de relaciones entre tablas mediante claves foráneas, lo que permite la creación de bases de datos complejas y flexibles.

Cabe señalar que MySQL utiliza un lenguaje de consulta estándar llamado SQL (Structured Query Language) para interactuar con las bases de datos. Con SQL, se pueden realizar operaciones como insertar, actualizar, eliminar y consultar datos en las tablas.

MySQL también ofrece una serie de herramientas y características avanzadas para garantizar la escalabilidad, disponibilidad y rendimiento. Entre ellas se encuentran:

* Replicación: permite la replicación de datos entre varios servidores, lo que permite una mejor escalabilidad y disponibilidad.
* Particionamiento: permite dividir una tabla en varias particiones para mejorar el rendimiento en consultas de gran tamaño.
* Transacciones: soporta transacciones para garantizar la integridad de los datos.
* Seguridad: ofrece varias opciones de seguridad para proteger los datos, como autenticación de usuarios y encriptación de datos.

MySQL es compatible con una amplia variedad de sistemas operativos y lenguajes de programación, incluyendo Windows, Linux, MacOS, y lenguajes como Java, PHP, Python y C#. Además, existe una gran cantidad de herramientas y librerías disponibles para facilitar el desarrollo y la administración de bases de datos MySQL.

## DLL Y DML

DLL (Data Definition Language) [9] es un subconjunto de SQL que se utiliza para definir la estructura de una base de datos, es decir, para crear y modificar tablas, esquemas, vistas, índices y otros objetos relacionados con la estructura de una base de datos. Algunos ejemplos de instrucciones DDL son:

* CREATE: para crear una tabla, vista, índice, procedimiento almacenado, etc.
* ALTER: para modificar una tabla, vista, índice, procedimiento almacenado, etc.
* DROP: para eliminar una tabla, vista, índice, procedimiento almacenado, etc.

DML (Data Manipulation Language) es otro subconjunto de SQL que se utiliza para interactuar con los datos almacenados en una base de datos, es decir, para insertar, actualizar y eliminar datos. Algunos ejemplos de instrucciones DML son:

* SELECT: para recuperar datos de una tabla.
* INSERT: para insertar nuevos datos en una tabla.
* UPDATE: para actualizar datos existentes en una tabla.
* DELETE: para eliminar datos de una tabla.

## IMPORTANCIA DE LOS ROLES EN UNA APLICACIÓN.

Los roles son [10] una característica importante en una aplicación ya que permiten controlar y restringir el acceso a los datos y funciones de la aplicación de manera granular. Al asignar roles a los usuarios, se pueden establecer permisos específicos para cada usuario, lo que ayuda a garantizar la seguridad de la aplicación y los datos.

Algunas de las ventajas de utilizar roles en una aplicación son:

* Seguridad: Al asignar roles específicos a los usuarios, se pueden restringir los permisos de acceso a los datos y funciones de la aplicación, lo que ayuda a prevenir el acceso no autorizado y a proteger los datos sensibles.
* Flexibilidad: Los roles permiten una gran flexibilidad en la asignación de permisos, lo que significa que se pueden crear roles específicos para diferentes funciones y niveles de acceso en la aplicación.
* Facilidad de administración: Al asignar roles a los usuarios, se pueden controlar los permisos de acceso de manera centralizada, lo que facilita la administración de los permisos en una aplicación.
* Mejora del rendimiento: Al restringir el acceso a los datos y funciones de la aplicación, se puede mejorar el rendimiento ya que solo se permitirá acceder a los datos necesarios para cada rol.
* Mayor audibilidad: Al asignar roles específicos a los usuarios, se puede rastrear y auditar las acciones de cada usuario, lo que ayuda a detectar posibles problemas de seguridad.

En resumen, los roles son una herramienta esencial en una aplicación ya que ayudan a garantizar la seguridad y flexibilidad en la gestión de los permisos de acceso, mejoran el rendimiento y facilitan la administración de permisos de una aplicación.

## MYSQL CONNECTOR Y JDBC.

MySQL Connector/J es un [11] driver JDBC (Java Database Connectivity) desarrollado por MySQL AB (ahora propiedad de Oracle) para conectar programas escritos en Java con una base de datos MySQL. JDBC es una API de Java que proporciona una interfaz estándar para conectarse a una base de datos desde un programa Java.

MySQL Connector/J es uno de los drivers JDBC disponibles para MySQL, y se utiliza para establecer una conexión a la base de datos, enviar consultas y recibir resultados. Es compatible con todas las versiones de MySQL desde la versión 3.23 y también es compatible con las últimas versiones de Java.

Para utilizar MySQL Connector/J en un programa Java, primero se debe incluir la biblioteca en el proyecto. Luego se puede utilizar la clase DriverManager para establecer una conexión a la base de datos. Una vez que se ha establecido la conexión, se puede utilizar la interfaz Connection para enviar consultas y recibir resultados.

Aquí hay un ejemplo de cómo utilizar MySQL Connector/J para conectarse a una base de datos:.

Texto

Descripción generada automáticamente

Figura 6. Diagrama básico de sockets.

En este ejemplo, se importa la biblioteca de JDBC de Java y se crea una clase llamada MySQLExample. En el método main, se registra el driver JDBC de MySQL Connector/J con Class.forName(JDBC\_DRIVER). Luego se establece una conexión a la base de datos utilizando la clase DriverManager y se almacena en una variable conn. El DriverManager necesita los parámetros de conexión como el url, user y password para establecer la conexión.

Una vez que se ha establecido la conexión, se cierra la conexión y la declaración con conn.close() y stmt.close() respectivamente.

En caso de que ocurra algún error en la conexión o en el proceso, se maneja con los bloques catch que imprimen el error en la consola.

Es importante notar que en este ejemplo no se realiza ninguna operación en la base de datos, solo se establece la conexión y se cierra.

## ROLES Y USUARIOS RELACIÓN BASE GENERAL.

La forma que generalmente se establece una relación entre una tabla llamada "usuario" y otra llamada "rol" para que cada usuario tenga asignado un rol es [12] mediante la creación de una clave foránea en la tabla "usuario" que haga referencia al campo "codigo\_rol" de la tabla "rol".

Por ejemplo, se podría crear las siguientes tablas:

**Tabla "usuario":**

* identificación (clave primaria)
* codigo\_rol (clave foránea hacia la tabla rol)
* codigo\_estado
* contraseña
* fecha\_creacion

**Tabla "rol":**

* codigo\_rol (clave primaria)
* nombre
* descripcion

En esta estructura, cada usuario tendría un campo "codigo\_rol" que hace referencia a un registro en la tabla "rol" mediante una clave foránea, el cual guarda un código, un nombre y una descripción del rol, permitiendo así asociar un rol específico a cada usuario.

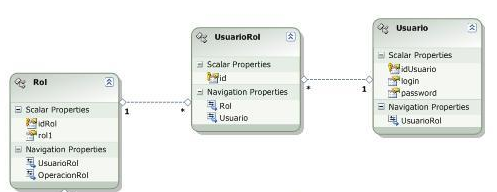


Figura 7. Código sencillo sobre el funcionamiento de timers.

En la figura anterior se puede acceder a esta relación, como se observa un usuario tiene un rol, por lo cual está conectado, el usuario estará relacionado y por tanto va a tener implícitamente un código de rol.

## PROGRAMACION ORIENTADA A OBJETOS (POO).

Programación Orientada a Objetos (POO, por sus siglas en inglés OOP, Object-Oriented Programming) [13] es un paradigma de programación que se basa en el concepto de "objetos", que representan entidades reales o virtuales con atributos y comportamientos.

En la programación orientada a objetos, los objetos interactúan entre sí a través de mensajes, y cada objeto es una instancia de una clase, que es una plantilla o modelo para la creación de objetos. Las clases pueden ser utilizadas para crear objetos similares y compartir atributos y comportamientos.

La programación orientada a objetos proporciona una abstracción y encapsulamiento de los datos y comportamientos de los objetos, lo que permite una mejor organización y reutilización del código. También promueve el modularidad y la separación de preocupaciones, lo que facilita la mantenibilidad y la escalabilidad del software.

Características de POO:

* **Abstracción:** Es un concepto clave que permite representar entidades reales o virtuales de manera simplificada y resumida. En la programación orientada a objetos, la abstracción se logra a través de la creación de clases, que son plantillas o modelos que describen los atributos y comportamientos de un objeto. Las clases permiten ocultar los detalles técnicos y proporcionar una representación más fácil y comprensible de las entidades.

La programación orientada a objetos se sabe que, de alguna manera, trata de "modelizar" los elementos del mundo real. En el mundo existe un universo de objetos que colaboran entre sí para realizar tareas de los sistemas. Llevado al entorno de la programación, también se debe programar una serie de clases a partir de las cuales se puedan instanciar objetos que colaboran entre sí para la resolución de problemas. Si se asume esto, a la vista de las situaciones que ocurren en el mundo real, se puede entender la abstracción.

Características:

* + *Simplificación*: Representa entidades reales o virtuales de manera más sencilla y resumida.
  + *Ocultamiento de detalles*: Oculta los detalles técnicos y permite una representación más fácil y comprensible de las entidades.
  + *Reutilización de código*: Permite la creación de clases que pueden ser reutilizadas en diferentes proyectos y situaciones.
  + *Mejor comprensión*: Permite una mejor comprensión y organización del código y las entidades representadas.
* **Encapsulamiento:** Se refiere a la protección de los datos y comportamientos de un objeto de su acceso externo. El encapsulamiento se logra a través de la creación de clases, que contienen atributos y métodos que describen los datos y comportamientos de un objeto. Estos atributos y métodos son accesibles solo dentro de la clase y no pueden ser modificados o accedidos desde fuera de la clase.

Características:

* + *Protección de datos*: Protege los datos y comportamientos de un objeto de su acceso externo, lo que promueve la seguridad y la integridad de los datos.
  + *Control de acceso*: Permite definir el nivel de acceso a los atributos y métodos de una clase, lo que facilita el control sobre la información.
  + *Mejor organización*: Permite una mejor organización y separación de preocupaciones, lo que facilita la mantenibilidad y escalabilidad del software.
  + *Modificación segura*: Protege los datos y comportamientos de un objeto de ser modificados por error o de manera accidental.
* **Herencia:** [14] Permite a los programadores crear nuevas clases a partir de clases existentes. La herencia permite que las propiedades y comportamientos de una clase sean heredados por una clase hija, lo que facilita la reutilización y la organización del código

Características:

* + *Composición:* Es cuando unos elementos están compuestos de otros, o que unos elementos están presentes en otros. Por ejemplo, el sistema respiratorio y los pulmones, la nariz, etc. Se puede decir que los pulmones están dentro del sistema respiratorio, así como dentro de los pulmones se encuentran bronquios y alvéolos. En esta jerarquía de elementos se tiene composición porque donde unos forman parte de otros. En una factura también se puede decir que puede haber una jerarquía de composición. La factura tiene un cliente, varios conceptos facturables, un impuesto, etc.
  + *Clasificación*: Este tipo de jerarquización indica que unos elementos son una especialización de otros. Por ejemplo, los animales, donde se tiene vertebrados e invertebrados. Luego, dentro de los vertebrados se encuentran aves, reptiles, mamíferos, etc. En los mamíferos se encuentran perros, vacas, conejos... Éste es el tipo de jerarquización en que quiero que te fijes.
  + *Reutilización de código*: Permite la reutilización de código y funcionalidades de una clase padre en una clase hija, lo que ahorra tiempo y esfuerzo al desarrollador.
  + *Organización de clases*: Permite la organización de clases en una jerarquía, lo que facilita la comprensión y la estructuración del software.
* **Polimorfismo:** Permite a los objetos de diferentes clases comportarse de manera similar aunque tengan diferentes implementaciones internas. El polimorfismo permite que un objeto de una clase pueda ser tratado como un objeto de otra clase si cumple con ciertos requisitos de tipo o interfaz.

Características:

* + *Flexibilidad*: Proporciona una mayor flexibilidad y adaptabilidad al software, permitiendo que objetos de diferentes clases se comporten de manera similar.
  + *Interfaces*: Permite definir una interfaz común para objetos de diferentes clases, lo que facilita la interoperabilidad y el intercambio de información entre objetos.

Imagen que contiene Diagrama

Descripción generada automáticamente

Figura 8. Características de POO.

Ventajas de POO:

* **Separación de responsabilidades**: Permite la separación de preocupaciones (SoC), ya que cada capa tiene una sola responsabilidad.
* **Fácil de desarrollar**: Este estilo arquitectónico es especialmente fácil de implementar, además de que es muy conocido y una gran mayoría de las aplicaciones la utilizan.
* **Fácil de probar**: Debido a que la aplicación construida por capas es posible ir probando de forma individual cada capa, lo que permite probar por separada cada capa.
* **Fácil de mantener**: Debido a que cada capa hace una tarea muy específica, es fácil detectar el origen de un bug para corregirlo, o simplemente se puede identificar donde se debe aplicar un cambio.
* **Seguridad:** La separación de capas permite el aislamiento de los servidores en subredes diferentes, lo que hace más difícil realizar ataques.

## 3.10 PROTOCOLO SIMPLE DE TRANSFERENCIA DE CORREO (SMTP).

Es un protocolo TCP/IP [15] que se utiliza para se usa para enviar correos desde un cliente de correo a un servidor de correo un cliente de correo puede ser de Gmail, Yahoo Outlook o un servidor de correo de un hosting. Normalmente se utiliza con POP3 o con el protocolo de acceso a mensajes de Internet (IMAP) para guardar mensajes en un buzón del servidor y descargarlos periódicamente del servidor para el usuario.

Imagen que contiene Texto

Descripción generada automáticamente}

Figura 9. Funcionamiento Correo electrónico.

Permite el intercambio de mensajes de correo electrónico entre servidores de correo. No es seguro, ya que no proporciona ninguna forma de autenticación o encriptación para la información transmitida y utiliza el puerto 25 para la transmisión de datos.

Es un estándar ampliamente aceptado y compatible con la mayoría de los sistemas de correo electrónico que es fácil de configurar y utilizar. También permite el envío de correos electrónicos desde cualquier lugar con acceso a Internet y además es escalable y puede manejar grandes cantidades de correos electrónicos sin interrupciones.

La siguiente tabla describe los mandatos de SMTP y sus funciones, e indica si el servidor SMPT.

Tabla 1. Mandatos de SMTP

|  |  |
| --- | --- |
| AUTH (Authentication) | Indica un mecanismo de autenticación al servidor SMTP. Se soportan tanto PLAIN como LOGIN. |
| DATA (Data) | Considera las líneas situadas después del mandato como correo electrónico del remitente. |
| EHLO (Extension Hello) | Habilita las extensiones SMTP. |
| EXPN (Expand) | Solicita al receptor que confirme que se ha identificado una lista de correo. |
| HELO (Hello) | Identifica al remitente de SMTP ante el receptor de SMTP. |
| HELP (Help) | Solicita al receptor que envíe información de ayuda al remitente. |
| MAIL (Mail) | Inicia una transacción de correo electrónico para entregar el correo electrónico a uno o más destinatarios. |
| NOOP (Noop) | Solicita al receptor que envíe una respuesta válida (pero sin especificar otras acciones). |
| QUIT (Quit) | Solicita al receptor que envíe una respuesta válida y luego cierre el canal de transmisión. |
| RCPT (Recipient) | Identifica un destinatario individual de correo electrónico. |
| RSET (Reset) | Finaliza la transacción de correo electrónico actual. |
| SAML (Send and mail) | Entrega correo electrónico a una o más estaciones de trabajo y destinatarios si el usuario no está activo. |
| SEND (Send) | Entrega correo electrónico a una o más estaciones de trabajo. |
| SOML (Send or mail) | Entrega correo electrónico a una o más estaciones de trabajo o destinatarios si el usuario no está activo. |
| STARTTLS (Start Transport Layer Security) | Solicita al servidor SMTP que inicie una negociación de capa de sockets seguros (SSL) o TLS con el cliente SMTP para establecer una sesión SSL o TLS. |
| TURN (Turn) | Solicita al receptor que envíe una respuesta válida y que, a continuación, se convierta en el remitente de SMTP, o bien solicita al receptor que envíe una respuesta de rechazo y siga siendo el receptor de SMTP. |
| VRFY (Verify) | Solicita al receptor que confirme que se ha identificado un usuario. |

## 3.11 PROTOCOLO POST OFFICE (POP).

Es un protocolo de correo electrónico [16] que permite a los usuarios descargar mensajes de correo electrónico desde un servidor remoto a sus computadoras locales. Una vez que el usuario se conecta al servidor de correo electrónico utilizando un cliente de correo electrónico compatible con POP, el servidor autentica al usuario y permite que los mensajes se descarguen a la computadora local del usuario. Los mensajes descargados se almacenan localmente y se pueden acceder sin conexión a Internet.

El protocolo POP 3 es la última versión del protocolo POP que se actualizó en 1988. Este protocolo se ha ido actualizando a lo largo del tiempo, pero ha seguido manteniendo su nombre POP3. Junto a IMAP, POP3 es el protocolo más utilizado en todo el mundo para conectar un cliente de correo a un servidor email.

Para conectarse a un servidor POP son necesarios varios datos:

* **Nombre del host**: Se trata del nombre o dirección IP del servidor de correo.
* **Nombre de usuario**: El nombre del usuario que normalmente coincide con la cuenta de correo electrónico.
* **Contraseña de usuario**: El password que utiliza el usuario para acceder a su correo electrónico.
* **Puerto de acceso**: El puerto de conexión que se utiliza para enviar el correo. Normalmente el protocolo POP3 utiliza el puerto 110 para conexiones no cifradas y el puerto 995 con conexiones TLS/SSL (cifradas).

También es importante conocer el tipo de cifrado que se utiliza para poder indicarlo en la configuración del cliente de correo electrónico que se utiliza para el acceso.

Características:

* Permite descargar mensajes desde un servidor remoto a una computadora local.
* No sincroniza automáticamente los mensajes enviados y recibidos entre el servidor y la computadora local.
* Solo permite acceder a los mensajes descargados mientras la computadora esté conectada a Internet.
* Algunas versiones de POP incluyen funciones adicionales, como la eliminación automática de mensajes después de descargarlos o la opción de dejar los mensajes en el servidor después de descargarlos.

## 3.12 IMAP.

IMAP (Protocolo de Mensajería de Acceso a Internet) es un protocolo de correo electrónico [17] que permite a los usuarios acceder y gestionar sus mensajes de correo electrónico almacenados en un servidor remoto. el cliente de correo electrónico se conecta al servidor de correo electrónico y permite al usuario acceder y gestionar sus mensajes de correo electrónico, incluyendo la opción de descargar mensajes a la computadora local, eliminar mensajes del servidor, etiquetar mensajes y mover mensajes entre carpetas. Los cambios realizados en la computadora local se reflejan en el servidor y viceversa, permitiendo una sincronización completa entre el servidor y la computadora local.

Simplemente es necesario contar con un equipo que tenga acceso a Internet y poder entrar a ese servidor. Este protocolo no permite enviar correos. Lo que hace realmente es dar acceso a los mensajes que hay almacenados en ese servidor. Tiene una ventaja frente a otros protocolos similares y es que IMAP permite visualizar el contenido en ese servidor sin necesidad de descargarlo. De esta forma, un usuario puede acceder desde un ordenador a un servidor, ver los mensajes y no tener que bajarlos a su propio equipo.

Fue creado en el año 1986 y llegó como sustituto del protocolo POP. Desde entonces han aparecido diferentes versiones que han mejorado el protocolo y han permitido nuevas funciones a los usuarios. La primera de ellas fue IMAP 2, que surgió solo un par de años después y ya fue distribuida de forma pública. Posteriormente aparecieron otras versiones como IMAP 3, IMAP2 bis o IMAP 4. Por tanto, se puede decir que gracias al protocolo IMAP los correos electrónicos se pueden sincronizar entre un equipo y un servidor. Permite crear diferentes carpetas y almacenarlos allí. Esto nos permite acceder al contenido guardado desde cualquier dispositivo que tenga acceso a Internet.

El protocolo IMAP utiliza el protocolo TCP sobre el protocolo IP, dependiendo del protocolo IMAP que se utiliza, los puertos TCP utilizados son diferentes. A continuación, puedes ver los puertos TCP más habituales que se usan:

* Puerto 143 TCP para IMAP.
* Puerto 220 TCP para IMAP3.
* Puerto 993 TCP para IMAPS, la versión segura del protocolo que nos permite añadir una capa de confidencialidad a todas las comunicaciones realizadas.

Los principales objetivos de utilizar IMAP son:

* Acceder a los correos desde cualquier equipo
* Compartir el buzón de correos con terceros
* Gestionar mejor los e-mails
* Liberar espacio en los dispositivos

IMAP y POP3 son los protocolos más utilizados para gestionar el envío y recepción de correos electrónicos entre cliente y servidor. Las principales diferencias entre ambos protocolos son:

* IMAP almacena los correos en el servidor mientras que POP3 los descarga en el cliente para almacenarlos de forma local. EL protocolo IMAP trabaja de forma directa sobre el servidor de correo electrónico, se muestra su contenido al usuario. En el caso del POP3 se conecta al servidor y descarga su contenido y lo almacena de forma local, pudiendo dejar una copia en el servidor si se configura para ello.
* Al utilizar POP3 se dispondrá del correo almacenado en el cliente, por lo que no es necesario disponer de conexión a internet para consultar los mensajes descargados. Con el protocolo IMAP se necesita de acceso a internet para poder acceder al servidor de correo y así poder leer los mensajes.
* El espacio necesario en el servidor para almacenar los correos será mucho mayor en IMAP que en POP3, ya que en este último se descargan y almacenan de forma local liberando el espacio del servidor. Esta característica es muy importante a la hora de configurar el correo, sobre todo en empresas o particulares que reciban una gran cantidad de correos al día y que pueden llenar fácilmente la capacidad de almacenaje que tienen en su servidor de correo.
* El correo IMAP es ideal para poder acceder a los mensajes desde múltiples dispositivos como ordenadores, smartphone o tablets. Al trabajar directamente en el servidor, cualquier cambio será reflejado en todos los dispositivos. Por su parte, el protocolo POP3 no admite conexiones simultáneas para acceder a los correos.
* Las carpetas creadas en IMAP no pueden ser leídas en POP3, por lo que es importante tener en cuenta este aspecto a la hora de elegir qué protocolo utilizar.

## 3.13 CIFRADO SSL Y TLS.

SSL es el acrónimo de Secure Sockets Layer (capa de sockets seguros), la tecnología estándar para mantener segura una conexión a Internet, así como para proteger cualquier información confidencial que se envía entre dos sistemas e impedir que los delincuentes lean y modifiquen cualquier dato que se transfiera, incluida información que pudiera considerarse personal. Los dos sistemas pueden ser un servidor y un cliente (por ejemplo, un sitio web de compras y un navegador) o de servidor a servidor (por ejemplo, una aplicación con información que puede identificarse como personal o con datos de nóminas).

Esto lo lleva a cabo asegurándose de que todos los datos que se transfieren entre usuarios y sitios web o entre dos sistemas sean imposibles de leer. Utiliza algoritmos de cifrado para codificar los datos que se transmiten e impedir que los hackers los lean al enviarlos a través de la conexión. Esta información podría ser cualquier dato confidencial o personal, por ejemplo, números de tarjeta de crédito y otros datos bancarios, nombres y direcciones.

El protocolo TLS (Transport Layer Security, seguridad de la capa de transporte) es solo una versión actualizada y más segura de SSL. Si bien aún se denominan a nuestros certificados de seguridad SSL porque es un término más común, al comprar certificados SSL en DigiCert, en realidad se compran los certificados TLS más actualizados con la opción de cifrado ECC, RSA o DSA.

SSL (Secure Sockets Layer) y TLS (Transport Layer Security) son protocolos de seguridad que se utilizan para proteger las comunicaciones en Internet. Ambos proporcionan autenticación de servidor, privacidad y integridad de los datos transmitidos a través de la red.

SSL fue el primer protocolo de seguridad utilizado en la World Wide Web, pero ha sido reemplazado por TLS debido a una serie de vulnerabilidades de seguridad identificadas en SSL. TLS es un protocolo más reciente y mejorado que soluciona estos problemas y ofrece mejores características de seguridad.

El funcionamiento de SSL/TLS es el siguiente:

* El cliente envía una solicitud de conexión a un servidor web seguro.
* El servidor responde con un certificado digital que incluye información sobre la identidad del servidor y una clave pública.
* El cliente verifica la identidad del servidor y, si está satisfecho, utiliza la clave pública para establecer un canal seguro de comunicación con el servidor.
* Durante la comunicación, los datos se cifran y se decifran mediante la clave pública y una clave privada que solo es conocida por el cliente y el servidor.

Las ventajas de utilizar SSL/TLS son las siguientes:

* Protección de los datos: los datos transmitidos a través de un canal SSL/TLS se cifran y son inaccesibles para cualquier tercero que intente interceptarlos.
* Autenticación del servidor: el cliente puede verificar la identidad del servidor antes de transmitir datos confidenciales.
* Integridad de los datos: SSL/TLS incluye un mecanismo de comprobación de integridad que garantiza que los datos no hayan sido modificados en tránsito.

Las principales diferencias entre SSL y TLS son las siguientes:

* TLS es más seguro que SSL, ya que incluye mejoras en la seguridad que solucionan las vulnerabilidades identificadas en SSL.
* TLS es compatible con una mayor variedad de algoritmos de cifrado y otras características de seguridad, mientras que SSL está limitado a un conjunto más pequeño.
* TLS es un protocolo más moderno y ampliamente utilizado que SSL, que ha sido reemplazado en gran medida por TLS.

## MÉTODOS.

Antes de realizar la practica presente es necesario conocer algunos métodos que son necesarios para entender los códigos y algunas definiciones que se utilizaran durante la práctica.

Tabla 2. Métodos generalmente utilizados en conexión de java y bases de datos.

|  |  |
| --- | --- |
| MÉTODO | DESCRIPCIÓN |
| DriverManager.getConnection() | Este método se utiliza para establecer una conexión con la base de datos MySQL. Se puede utilizar con una cadena de conexión JDBC para especificar los detalles de la conexión, como el nombre de usuario y la contraseña. |
| Statement | Esta interfaz se utiliza para enviar sentencias SQL estáticas a la base de datos. Se puede utilizar para realizar operaciones de SELECT, INSERT, UPDATE y DELETE. |
| PreparedStatement | Esta interfaz se utiliza para enviar sentencias SQL parametrizadas a la base de datos. Es útil para evitar inyecciones SQL al usar parámetros en lugar de insertar directamente valores en la sentencia SQL. |
| ResultSet | Esta interfaz se utiliza para almacenar el resultado de una consulta SELECT. Se puede utilizar para recorrer los resultados de la consulta y recuperar los valores de cada columna. |
| com.mysql.jdbc.Driver | Esta clase se utiliza para cargar el controlador JDBC de MySQL. Se debe importar y registrar en el sistema antes de utilizar el DriverManager para establecer una conexión con la base de datos. |
| java.sql | Este paquete contiene las clases y interfaces necesarias para trabajar con bases de datos en Java, incluyendo las mencionadas anteriormente. |
| SessionFactory.openSession() | Este método se utiliza para abrir una sesión con la base de datos. Es necesario para llevar a cabo cualquier operación CRUD con la base de datos. |
| Session.save() | Este método se utiliza para guardar un objeto en la base de datos. Es equivalente a una operación INSERT en SQL. |
| Session.update() | Este método se utiliza para actualizar un objeto en la base de datos. Es equivalente a una operación UPDATE en SQL. |
| Session.delete() | Este método se utiliza para eliminar un objeto de la base de datos. Es equivalente a una operación DELETE en SQL. |
| Session.get() | Este método se utiliza para recuperar un objeto de la base de datos a partir de su clave primaria. Es equivalente a una operación SELECT en SQL. |
| org.hibernate.SessionFactory | Esta clase es una fábrica para las sesiones de Hibernate. Es necesaria para abrir sesiones y realizar operaciones con la base de datos. |
| org.hibernate.Session | Esta clase representa una sesión con la base de datos. Es necesaria para llevar a cabo cualquier operación CRUD con la base de datos. |

## TÉRMINOS.

Además, es necesario conocer el significado de una cantidad de términos un tanto complicados relacionados con la informática en cuanto a la conexión de una base de datos con java ee.

Tabla 3. Términos

|  |  |
| --- | --- |
| ORM (Object-Relational Mapping) | Una técnica que permite a un programador trabajar con objetos en lugar de con registros de una base de datos, haciendo que sea más fácil manipular y trabajar con datos en una aplicación. |
| JPA (Java Persistence API) | Una interfaz estándar para la implementación de ORM en aplicaciones Java EE. |
| Entity | Clases que representan las tablas de una base de datos en una aplicación JPA. |
| Persistence Unit | Un conjunto de entidades y configuraciones de una aplicación JPA. |
| EntityManager | Una interfaz para interactuar con entidades en una aplicación JPA. |
| JPQL (Java Persistence Query Language) | Un lenguaje de consulta similar a SQL para interactuar con entidades en una aplicación JPA. |
| Hibernate | Una popular implementación de ORM para aplicaciones Java. |
| MySQL Connector | Un driver que permite conectarse a una base de datos MySQL desde una aplicación Java. |
| DataSource | Un pool de conexiones a una base de datos que puede ser utilizado por una aplicación. |
| CRUD (Create, Read, Update, Delete) | Las operaciones básicas para interactuar con una base de datos, crear, leer, actualizar y eliminar registros. |
| Transacciones | Un conjunto de operaciones que se realizan en una base de datos como una unidad, ya sea todas se completen o ninguna. |
| DAO (Data Access Object) | Una clase o interfaz que proporciona métodos para interactuar con una base de datos. |
| JDBC (Java Database Connectivity) | Una API estándar para interactuar con bases de datos en aplicaciones Java. |
| Prepared Statement | Una sentencia SQL que ha sido precompilada para mejorar el rendimiento en aplicaciones JDBC. |
| ResultSet | Un conjunto de resultados obtenidos de una consulta a una base de datos en una aplicación JDBC. |
| Login | Proceso de verificación de credenciales de un usuario para acceder a un sistema o aplicación. |
| Session | Es un objeto que representa una conexión a la base de datos. Se utiliza para realizar operaciones CRUD en las entidades y manejar transacciones. |
| Persist | Es una operación que se utiliza para insertar una nueva entidad en la base de datos. |
| Merge | Es una operación que se utiliza para actualizar una entidad existente en la base de datos. |
| Remove | Es una operación que se utiliza para eliminar una entidad de la base de datos. |
| Find | Es una operación que se utiliza para recuperar una entidad de la base de datos a partir de su identificador. |
| Query | Es un objeto que se utiliza para ejecutar una consulta en la base de datos y recuperar una lista de resultados. |
| Criteria API | Es una interfaz de programación de consultas que se utiliza para construir consultas de forma dinámica. |
| JPQL | Es un lenguaje de consulta similar a SQL que se utiliza para realizar consultas en las entidades. |
| NamedQuery | Es una consulta que se define en la entidad mediante anotaciones o xml y se puede ejecutar mediante su nombre. |
| NativeQuery | Es una consulta que se ejecuta directamente en el motor de base de datos utilizando SQL nativo. |

# 4. DESARROLLO

# 4.1 CREACIÓN DEL PROYECTO

## 4.1.1 CREAR UNA APLICACIÓN LLAMADA LOGIN\_NETBEANS\_GRUPO#.

Abra el IDE de desarrollo Apache NetBeans.



Figura . Logotipo identificativo de Apache Netbeans

Primero dentro del disco C y cree una carpeta llamada DISTRIBUIDAS allí crea otra subcarpeta llamada NETBEANS, dentro de esta crea otra subcarpeta con el nombre de LOGIN\_NETBEANS\_GRUPO# a su vez dentro de esta última crea 2 carpetas, una con el nombre de “aplicativo” y otra con el nombre de “documentación”. Una vez haya creado las carpetas abra apache netbeans y seleccionar crear nuevo proyecto, en el tipo de aplicación busca Aplicación web

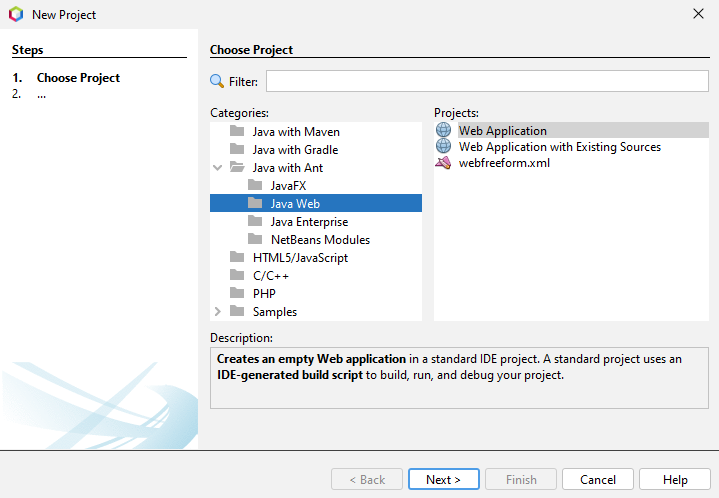


Figura . Selección del tipo de aplicación a desarrollar

Le da el siguiente nombre NETBEANS\_LOGIN\_GRUPO#, y en la opción que dice localización del proyecto presiona buscar y selecciona la carpeta “aplicativo” que fue creada con anterioridad.

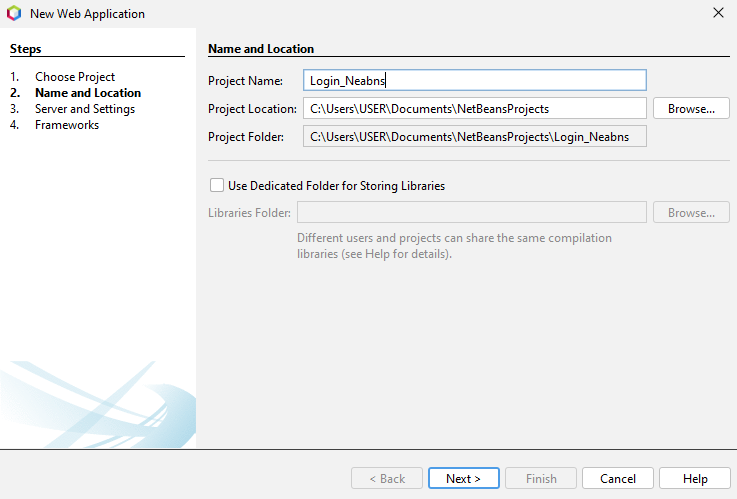


Figura . Selección del nombre y directorio donde guardar el proyecto.

De clic en crear y Verifique que la estructura del proyecto se encuentre de la siguiente manera.

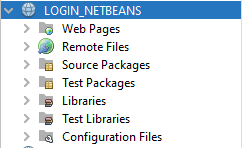


Figura . Estructura del proyecto generado.

## 4.1.2 CREACIÓN DE ESTRUCTURA MVC DEL PROYECTO.

Para aplicar la estructura de MVC en un proyecto es necesario dividir los componentes en secciones que contengan archivos o elementos similares y que cumplen una función acorde a cada capa del modelo MVC, en este caso, para el proyecto se ha decidido agruparlos en paquetes.

Para ello de clic derecho sobre Source Packeges, selecciona agregar y elije nuevo paquete.

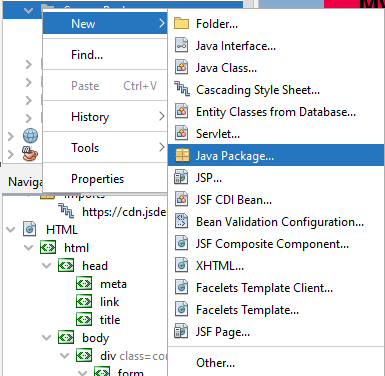


Figura . Creación de carpeta

Deberá crear 2 paquetes, con los nombres: modelo y controlador, no se creará un paquete vista debido a que en aplicaciones web la vista son las páginas web que suelen encontrarse en la carpeta web Pages. De preferencia nombre a las carpetas con la primera letra en mayúscula. Al final la estructura resultante deberá ser la siguiente.



Figura . Estructura del proyecto.

## 4.1.3 IMPORTACIÓN DE BIBLIOTECAS NECESARIAS.

Como se indicó anteriormente para la conexión a la base de datos en este caso se deberá constar de 3 librerías las cuales permitirán poder conectar nuestra aplicación con otras bases de datos. Para ello debe dar clic sobre la carpeta librerías, y seleccionar la opción añadir JAR.

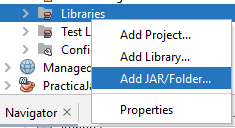


Figura . Creación de carpeta

Busca los archivos JAR de la siguiente figura, se seleccionan los tres y se importan.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Importación de librerías

## 4.1.4 CONEXIÓN CON LA BASE DE DATOS.

Para realizar la conexión a la base de datos se debe dirigir a la pestaña Services donde se debe seleccionar New Connection.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Creación de una nueva conexión

Selecciona el driver de Payara que aparece y presiona siguiente.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Driver de conexión

Luego se debe completar los campos con la información de la base de datos, como su nombre, el usuario, la contraseña, el puerto, etc. En este caso se tuvo que añadir a la URL “?serverTimezone=UTC” para poder conectarla.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Configuración de conexión

Presiona siguiente hasta finalizar y se obtiene la conexión de la siguiente forma.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Figura . Conexión con la BD

# 4.2 CREACIÓN DE VISTAS DEL PROYECTO

### 4.2.1 CREACIÓN DE ARCHIVOS JSF.

Con la base de datos conectada a Netbeans lo único que se debe hacer es generar las vistas automáticamente. Para esto se debe dirigir hacia Source Packages y seleccionar “Otro”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Creación de Vistas.

Proceda a buscar JSF y seleccione la opción JSF Pages From Entity Classes.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Selección de tipo de archivo

Se desplegará la siguiente vista donde se debe pasar todas las clases a la columna de la derecha.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de clases.

En la ventana siguiente se debe seleccionar al paquete de Session Bean al del modeo y al paquete JSF Classes al controlador. De esta manera, además de generar las vistas, también se generarán los controladores y algunos modelos automáticamente.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Configuración de paquetes JSF

Una vez terminado se da click en Finalizar y el esquema del proyecto debe ser el siguiente.

Texto

Descripción generada automáticamente con confianza baja

Figura . Esquema del proyecto con las vistas creada

### 4.2.2 CODIFICACIÓN DE ARCHIVO INDEX

El archivo índex será el encargado de ejecutar la aplicación y por tanto generar la pantalla para el inicio de sesión, por lo cual deberá crear un archivo que contenga un formulario, que llame al controlador de la aplicación que será creado más.

Texto

Descripción generada automáticamente

Figura . Codificación del Login en index.xhtml.

Modifique el código del archivo con la siguiente tabla.

Tabla . Codificación del archivo index

<?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:p="http://primefaces.org/ui"

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

<h:head>

<!--<link rel="stylesheet" type="text/css" href="resources/css/main.css"/>-->

<!--<link rel="stylesheet" type="text/css" href="resources/css/util.css"/>-->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous"></link>

<link rel="stylesheet" type="text/css" href="resources/css/login.css"/>

<title>Facelet Login</title>

</h:head>

<h:body>

<div class="container">

<div id="divForm" style="margin-left: 10%;">

<div id="formGroup">

<h:form id="formLogin">

<div class="imagenUsuario">

<img src="resources/img/user.png" />

</div>

<h3 class="mb-4 pb-1 fs-2 titulo-login">Inicio de Sesión</h3>

<p:growl id="growl" showDetail="true" />

<h:outputLabel value="Correo" for="peempEmplePK\_peempCodigo" class="text-start label"/>

<p:inputText class="input100 mb-3" placeholder="Ingrese su correo" id="peempEmail" value="#{xeusuUsuarController.selected.peempEmple.peempEmail}" required="true"/>

<h:outputLabel value="Contraseña" for="peempEmplePK\_peempCodigo" class="text-start label"/>

<p:password class="input100 mb-3" feedback="false" id="xeusuPaswd" value="#{xeusuUsuarController.selected.xeusuPaswd}" required="true" placeholder="Ingrese su contraseña"/>

<span class="focus-input100"></span>

<h:commandButton class="btn verde aumento" value="Ingresar" action="#{xeusuUsuarController.validar()}" />

</h:form>

<br/>

<h:form id="FormResetPassword">

<span class="txt1 aumento">

<h:commandLink value="¿Has olvidado tu contraseña?" action="resetearPassword" />

</span>

</h:form>

<h:form>

<h:commandButton class="btn registro aumento" value="¿No tiene cuenta? Regístrese" actionListener="peempEmple/CrearInvitado.xhtml" action="/peempEmple/CrearInvitado?faces-redirect=true"/>

</h:form>

</div>

</div>

<div id="imgLeft" style="margin-left: -47%; z-index: -1; width: 100vw !important; " >

<p:graphicImage id="imagenPrincipal" value="resources/img/monsterUniversity.jpg" library="poseidon-layout" />

</div>

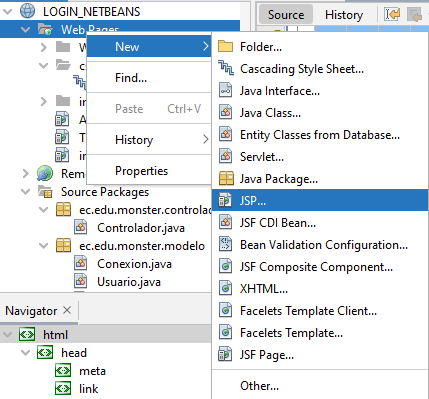
</div>

</h:body>

</html>

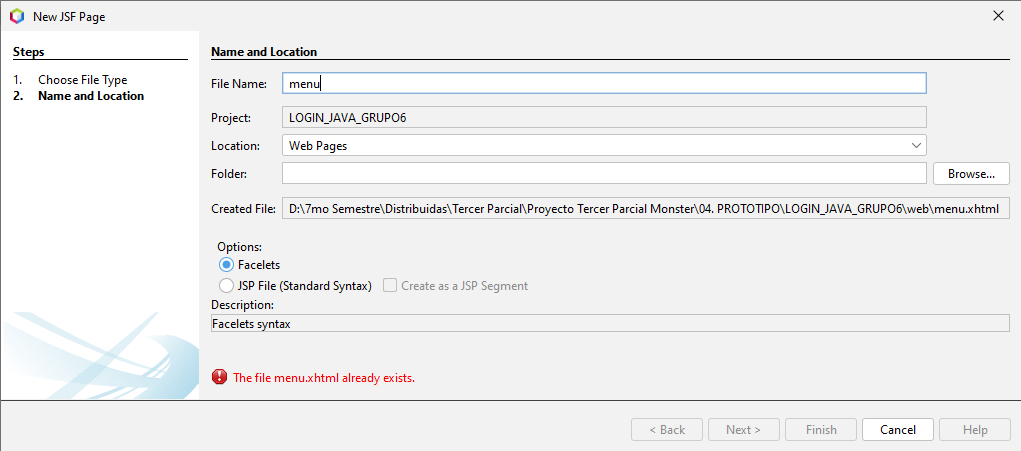
4.2.3 CREACIÓN DE ARCHIVO MENU.

Una de las ventanas más importantes es el menú, en este se mostrará la información del usuario, así como los menús a los distintos cruds y funcionalidades del programa, para ello se debe dirigir a Web\_Pages y seleccionar JSP.



*Figura 28. Cambio de nombre del archivo.*

Cree un archivo bajo el nombre “menú” y luego se debe seleccionar finalizar**.**



*Figura 29. Creación archivo menú.xhtml*

Coloque en el mismo el siguiente código.

<?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:p="http://primefaces.org/ui"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

<h:head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<link href="resources/css/cssLayout.css" rel="stylesheet" type="text/css"/>

<link href="resources/css/default.css" rel="stylesheet" type="text/css"/>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous"></link>

<title>Facelet Title</title>

<f:event type="preRenderView" listener="#{peempEmpleController.verificarSesion()}"/>

</h:head>

<h:body>

<ui:include src="/WEB-INF/jsfFragment/menu.xhtml" />

<div class="container-fluid">

<div class="row text-white" >

<div class="col-3 bg-dark " style=" height:90vh; text-align: center">

<h2 class="mt-5 pt-2">Usuario</h2>

<p:growl id="growl" showDetail="true" life="3000" />

<h:outputText value="Invitado" rendered="#{empty xeusuUsuarController.selected.xeuxpUsupeList}"/>

<ui:repeat value="#{xeusuUsuarController.selected.xeuxpUsupeList}" var="item" varStatus="estado">

<h:outputLabel value="#{item.xeperPerfi.xeperDescri}"/>

Tabla . Codificación del archivo menú.xhtml

</ui:repeat>

<br/><br/>

<p:graphicImage name="/img/#{xeusuUsuarController.obtenerNameFoto()}" style="width: 45%; height: 20vh; border-radius: 50%"/>

<br/>

<h:outputLabel value="" />

<br/><br/>

<h:outputLabel value="#{xeusuUsuarController.selected.peempEmple.peempNombre} #{xeusuUsuarController.selected.peempEmple.peempApellpaterno}"/>

<br/><br/>

<h:outputLabel value="#{xeusuUsuarController.selected.peempEmple.peempEmail}"/>

<br/><br/><br/><br/>

<h:form>

<p:commandButton value="Editar Perfil" actionListener="peempEmple/EditarEmpleado.xhtml" action="/peempEmple/EditarEmpleado?faces-redirect=true"/>

<br/><br/>

<p:commandButton value="Cerrar Sesión" actionListener="#{menuControlador.cerrarSesion()}" action="/index?faces-redirect=true"/>

</h:form>

<!--<p:commandLink value="Editar"/>

<br/><br/>

<p:commandLink value="Salir"/>-->

</div>

<div class="col-9" style="height:90vh;">

<p:graphicImage value="/resources/img/fondo.jpg" style="width: 80%; height: 80vh;"/>

</div>

</div>

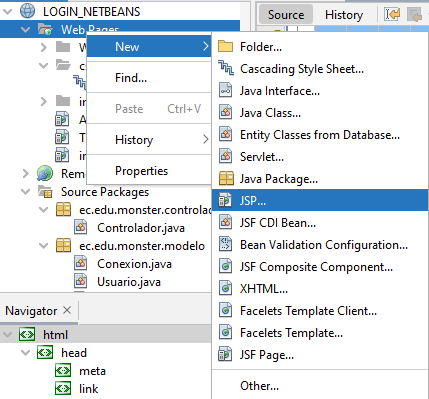
</div>

</h:body>

</html>

4.2.4 CREACIÓN DE ARCHIVO CAMBIARPASSWORD.

Uno de los requerimientos de la aplicación es poder cambiar la contraseña si es la primera vez que un usuario inicia sesión o si resetea su contraseña, para hacer esto se crea un nuevo archivo JSP Page.



*Figura 30. Selección nuevo JSP Pages.*

Cree un archivo bajo el nombre “cambiarPassword” y luego se debe seleccionar finalizar**.**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*Figura 31. Creación archivo cambiarPassword.xhtml*

Coloque en el mismo el siguiente código.

Tabla . Codificación del archivo cambiarPassword.xhtml

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

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

xmlns:p="http://primefaces.org/ui"

xmlns:f="http://xmlns.jcp.org/jsf/core">

<h:head>

<title>Facelet Title</title>

</h:head>

<h:body style="width: 95%;margin: auto auto;">

<p:panel style="width: 400px;margin: 0 auto; text-align: center;">

<h:form>

<p:growl id="growl" showDetail="true" />

<p:growl id="msgPass" showDetail="true" />

<p:panelGrid columns="2" id="pGrid" class="cuadro">

<p:outputLabel value="Contraseña Nueva" for="passNu"/>

<p:password required="true" id="passNu" value="#{xeusuUsuarController.newPassword}">

<f:validator validatorId="ContraseniaValidator"/>

<p:ajax update="msgPass" event="blur" />

</p:password>

<p:outputLabel value="Repetir Contraseña Nueva" for="repPassNu"/>

<p:password required="true" id="repPassNu" value="#{xeusuUsuarController.repPassword}"/>

<p:column />

<p:commandButton value="Cambiar contraseña" action="#{xeusuUsuarController.cambiarPassword()}" update="growl"/>

</p:panelGrid>

</h:form>

</p:panel>

</h:body>

</html>

4.2.5 CREACIÓN DE ARCHIVO RESETEARPASSWORD.

Al igual que la vista anterior, se necesita una pantalla para poder enviar el correo con la contraseña reseteada, para ello se crea la siguiente vista de la misma forma seleccionando un nuevo JSP Page.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

*Figura 32. Selección nueva JSP Pages.*

Cree un archivo bajo el nombre “resetarPassword” y luego se debe seleccionar finalizar**.**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*Figura 33. Creación archivo resetarPassword.xhtml*

Coloque en el mismo el siguiente código.

Tabla . Codificación del archivo resetarPassword.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://xmlns.jcp.org/jsf/html"

xmlns:p="http://primefaces.org/ui">

<h:head>

<title>Facelet Title</title>

</h:head>

<h:body style="width: 95%;margin: auto auto;">

<p:panel style="width: 400px;margin: 0 auto; text-align: center;">

<h:form>

<p:growl id="growl" life="3000"/>

<p:panelGrid columns="2" id="pGrid" class="cuadro">

<p:outputLabel value="Correo:" for="correo"/>

<p:inputText id="correo" value="#{xeusuUsuarController.selected.peempEmple.peempEmail}" required="true" />

<p:column />

<p:commandButton action="#{xeusuUsuarController.resetearPassword()}" value="Cambiar contraseña" />

</p:panelGrid>

</h:form>

</p:panel>

</h:body>

</html>

<li class="nav-item">

<a class="nav-link" href="#">Features</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Pricing</a>

</li>

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Dropdown link

</a>

<div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">

<a class="dropdown-item" href="#">Action</a>

<a class="dropdown-item" href="index.jsp">Salir</a>

</div>

</li>

</ul>

</div>

</nav>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

</body>

</html>

4.2.6 MODIFICACIÓN DE CRUDS.

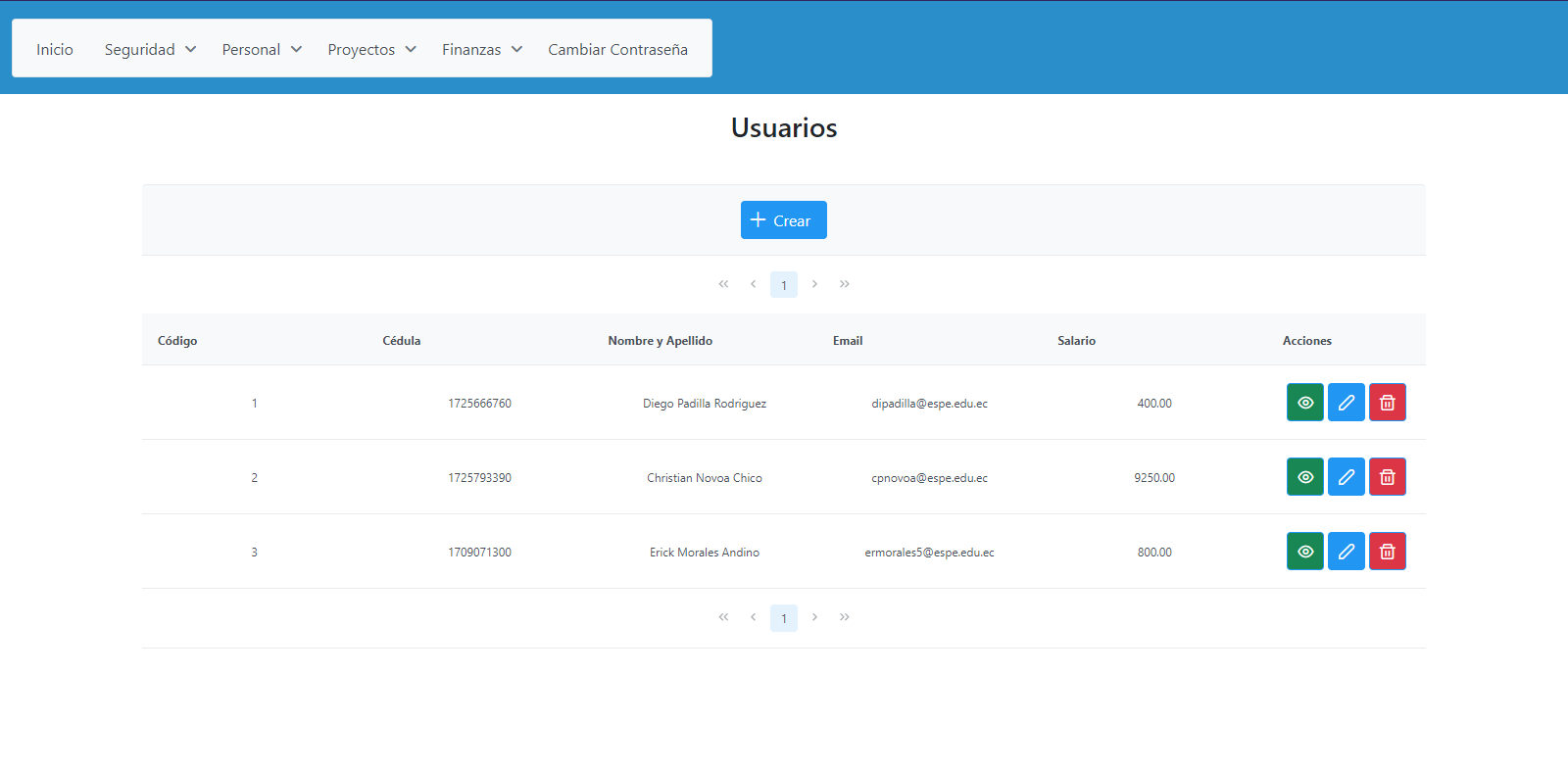
Gracias a la creación de vistas anteriormente realizado ya se tiene cruds implementados, sin embargo, para darle estilo y funcionalidad a los mismo, para ello se mostrará la aplicación de estilos en el archivo peempEmple/List.xhtml. Dentro de cada archivo crud se debe agregar las etiquetas de Bootstrap para poder importar sus estilos. A continuación se muestra el formulario con los estilos originales.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Sitio web

Descripción generada automáticamente

*Figura 34. CRUD original.*

Aplicando el código que se presenta en la tabla a continuación los cruds deberán tener un aspecto similar al siguiente, cada uno con las llamas a los controladores correspondientes que se crearán más adelante**.**



*Figura 35. CRUD modificado*

Tabla . Codificación del archivo peempEmple/List.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:ui="http://xmlns.jcp.org/jsf/facelets"

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

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:p="http://primefaces.org/ui">

<h:head>

<link rel="shortcut icon" href="#"></link>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/materialize/1.0.0/css/materialize.min.css"/>

<link rel="stylesheet" href="style.css"/>

<script type="text/javascript" src="https://kit.fontawesome.com/832497f32a.js"/>

</h:head>

<ui:composition template="/template.xhtml">

<ui:define name="title">

<ui:include src="/WEB-INF/jsfFragment/menu.xhtml" />

</ui:define>

<ui:define name="body">

<center>

<h3 class="mt-3">Usuarios</h3>

<h:panelGroup id="messagePanel" layout="block">

</h:panelGroup>

<h:form id="formAcceso" style="width: 85%">

<center>

<p:growl id="growl" showDetail="true" life="3000" />

</center>

<p:spacer height="3px"/>

<p:dataTable id="tablaAccesos" var="item" style="padding-left: 25px; padding-right: 25px; padding-top: 10px;padding-bottom: 32px; font-size: 12px " value="#{peempEmpleController.items}"

widgetVar="tablaAccesos" emptyMessage="No hay acceso bajo ese criterio" filteredValue="" editable="true" paginator="true" rows="10">

<f:facet name="header">

<center>

<p:commandButton value="Crear" icon="pi pi-plus" oncomplete="PF('dlg2').show();" actionListener="#{peempEmpleController.inicializarEmpleado}"/>

</center>

</f:facet>

<p:column headerText="Código" filterMatchMode="contains">

<center>

<h:outputText value="#{item.peempCurp}"/>

</center>

</p:column>

<p:column headerText="Cédula" filterMatchMode="contains">

<center>

<h:outputText value="#{item.peempCedula}"/>

</center>

</p:column>

</p:column>

</p:dataTable>

<!-- DIALOG CONFIRMACION -->

<p:confirmDialog global="true" showEffect="fade" hideEffect="fade" width="350">

<p:commandButton value="No" type="button" styleClass="ui-confirmdialog-no ui-button-flat"/>

<p:commandButton value="Si" type="button" styleClass="ui-confirmdialog-yes" />

</p:confirmDialog>

</h:form>

<!-- CREAR USUARIO -->

<ui:include src="Create.xhtml" />

<!-- EDITAR USUARIO -->

<ui:include src="Edit.xhtml" />

<!-- Ver USUARIO -->

<ui:include src="View.xhtml" />

</center>

</ui:define>

</ui:composition>

</html>

4.2.7 IMPLEMENTACIÓN DE TABVIEWS.

Para mostrar la información de forma más organizada se puede optar por utilizar TabViews dentro de los formularios, esto permite tener dividido y organizado lo que el usuario puede observar, en este caso se aplicarán a la lista de Empleados / Usuarios tanto al visualizar uno de estos como al Editar o Crear uno nuevo. Este proceso se repite por lo que se muestra a continuación el proceso de implementación en el formulario View y luego el código de los formularios Crear y Editar.

Para esto se abre el archivo ubicado en peempEmple/View.xhtml.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

*Figura 36. Ubicación del archivo.*

Dentro del mismo lo que se debe hacer es crear un <h:panelGrid donde irán los TabView ubicados, paso siguiente se coloca la etiqueta de <p:tabView>. Dentro de esta ya se podrá usar <p:tab title=" "> segmentando lo que se coloca en cada una, como se puede ver en la siguiente figura**.**



*Figura 37. Implementación de los TabView*

El resultado de implementarlos queda de la siguiente manera dentro del formulario.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

*Figura 38. Vista de los formularios con TabView*

A continuación se presenta el código del View, Crear y Editar con TabViews.

Tabla . Codificación del archivo peempEmple/ View.xhtml

<ui:composition

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

xmlns:f="http://java.sun.com/jsf/core"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:p="http://primefaces.org/ui"

xmlns:ui="http://java.sun.com/jsf/facelets">

<h:form enctype="multipart/form-data" id="formViewUsuario">

<p:dialog header="Ver Usuario" widgetVar="digViewAcceso" modal="true" height="100%" style="font-size: 14px" id="digViewAcceso">

<p:panelGrid columns="2" style="width: 100%">

<h:panelGrid columns="2" cellpadding="7px">

<h:outputLabel value="Nombre:" for="peempNombre" />

<h:outputLabel id="peempNombre" value="#{peempEmpleController.selected.peempNombre}" title="#{bundle.CreatePeempEmpleTitle\_peempNombre}" />

<h:outputLabel value="Apellido Paterno:" for="peempApellpaterno" />

<h:outputLabel id="peempApellpaterno" value="#{peempEmpleController.selected.peempApellpaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellpaterno}" />

<h:outputLabel value="Apellido Materno:" for="peempApellmaterno" />

<h:outputLabel id="peempApellmaterno" value="#{peempEmpleController.selected.peempApellmaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellmaterno}" />

<h:outputLabel value="Cédula:" for="peempCedula" />

<h:outputLabel id="peempCedula" value="#{peempEmpleController.selected.peempCedula}" title="#{bundle.CreatePeempEmpleTitle\_peempCedula}" />

</h:panelGrid>

<p:panelGrid columns="1" rendered="#{peempEmpleController.selected != null}" id="Image" style="text-align: center" >

<p:graphicImage name="/img/#{peempEmpleController.selected.peempFoto}" width="100px" height="100px" rendered="#{peempEmpleController.selected.peempFoto ne 'NULL' }" />

</p:panelGrid>

</p:panelGrid>

<h:panelGrid columns="1" style="width: 100%">

<p:tabView>

<p:tab title="Información Personal">

<h:panelGrid columns="2" border="0" cellpadding="5px">

<h:outputLabel value="Sexo:" for="pesexCodigo" />

<h:outputLabel id="pesexCodigo" value="#{peempEmpleController.selected.pesexCodigo.pesexNomb}" title="#{bundle.CreatePeempEmpleTitle\_pesexCodigo}" />

<h:outputLabel value="Género:" for="codgenero" />

<h:outputLabel id="codgenero" value="#{peempEmpleController.selected.codgenero.descripcion}" title="#{bundle.CreatePeempEmpleTitle\_codgenero}" />

<h:outputLabel value="Fecha de Nacimiento:" for="peempFecnac" />

<h:outputLabel id="peempFecnac" value="#{peempEmpleController.selected.peempFecnacimiento}" title="#{bundle.CreatePeempEmpleTitle\_peempFecnacimiento}">

<f:convertDateTime pattern="MM/dd/yyyy" />

</h:outputLabel>

<h:outputLabel value="Estado Civil:" for="peescCodigo" />

<h:outputLabel id="peescCodigo" value="#{peempEmpleController.selected.peescCodigo.peescNomb}" title="#{bundle.CreatePeempEmpleTitle\_peescCodigo}" />

<h:outputLabel value="Nacionalidad:" for="codignac" />

<h:outputLabel id="codignac" value="#{peempEmpleController.selected.codnacion.descrinac}" title="#{bundle.CreatePeempEmpleTitle\_codnacion}" />

</h:panelGrid>

</p:tab>

<p:tab title="Información Laboral">

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="Salario:" for="peempSalario" />

<h:outputLabel id="peempSalario" value="$#{peempEmpleController.selected.peempSalario}" title="#{bundle.CreatePeempEmpleTitle\_peempSalario}" />

<h:outputLabel value="Departamento:" for="coddepart" />

<h:outputLabel id="coddepart" value="#{peempEmpleController.selected.coddepart.nombredepto}" title="#{bundle.CreatePeempEmpleTitle\_coddepart}" />

</h:panelGrid>

</p:tab>

</p:tabView>

</h:panelGrid>

</p:dialog>

</h:form>

</ui:composition>

Tabla . Codificación del archivo peempEmple/ Create.xhtml

<ui:composition

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

xmlns:f="http://java.sun.com/jsf/core"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:p="http://primefaces.org/ui"

xmlns:ui="http://java.sun.com/jsf/facelets">

<h:form enctype="multipart/form-data">

<f:event listener="#{peempEmpleController.limpiar}" type="preRenderComponent"/>

<p:dialog header="Crear Usuario" widgetVar="dlg2" modal="true" width="55%" height="75vh" style="font-size: 30px" >

<p:growl id="msgCedula" showDetail="true" life="7000" />

<p:panelGrid columns="2" style="font-size: 14px" >

<h:panelGrid columns="2" cellpadding="5px" >

<h:outputLabel for="peempCedula" value="Cédula:\*"/>

<h:inputText id="peempCedula" value="#{peempEmpleController.selected.peempCedula}">

<f:validator validatorId="cedulaValidator"/>

<p:ajax update="msgCedula" event="blur" />

</h:inputText>

<h:outputLabel value="Nombre:\*" for="peempNombre" />

<h:inputText id="peempNombre" value="#{peempEmpleController.selected.peempNombre}" title="#{bundle.CreatePeempEmpleTitle\_peempNombre}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempNombre}"/>

<h:outputLabel value="Apellido Paterno:\*" for="peempApellpaterno" />

<h:inputText id="peempApellpaterno" value="#{peempEmpleController.selected.peempApellpaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellpaterno}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempApellpaterno}"/>

<h:outputLabel value="Apellido Materno:\*" for="peempApellmaterno" />

<h:inputText id="peempApellmaterno" value="#{peempEmpleController.selected.peempApellmaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellmaterno}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_ppeempApellmaterno}"/>

</h:panelGrid>

</p:panelGrid>

<h:panelGrid columns="1" style="width: 100%;font-size: 14px">

<p:tabView>

<p:tab title="Información Personal">

<h:panelGrid columns="2" border="0" cellpadding="5px" style="font-size: 14px">

<h:outputLabel value="Sexo:\*" for="pesexCodigo" />

<h:selectOneMenu id="pesexCodigo" value="#{peempEmpleController.selected.pesexCodigo}" title="#{bundle.CreatePeempEmpleTitle\_pesexCodigo}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_pesexCodigo}">

<f:selectItems value="#{pesexSexoController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

<h:outputLabel value="Género:\*" for="codgenero" />

<h:selectOneMenu id="codgenero" value="#{peempEmpleController.selected.codgenero}" title="#{bundle.CreatePeempEmpleTitle\_codgenero}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_codgenero}">

<f:selectItems value="#{pegenGenerController.itemsAvailableSelectOne}"/>

<h:outputLabel value="Estado Civil:\*" for="peescCodigo" />

<h:selectOneMenu id="peescCodigo" value="#{peempEmpleController.selected.peescCodigo}" title="#{bundle.CreatePeempEmpleTitle\_peescCodigo}" >

<f:selectItems value="#{peestEstcivController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

<h:outputLabel value="Nacionalidad:\*" for="codnacion" />

<h:selectOneMenu id="codnacion" value="#{peempEmpleController.selected.codnacion}" title="#{bundle.CreatePeempEmpleTitle\_codnacion}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_codnacion}">

<f:selectItems value="#{penacNacionController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

</h:panelGrid>

</p:tab>

<p:tab title="Información de Contacto">

<p:growl id="msgEmail" showDetail="true" life="7000" />

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="País:\*" for="pepaPai" />

<h:selectOneMenu value="#{peempEmpleController.codPais}" style="display: block;width:250px;margin-top: 10px;">

<f:selectItem itemLabel="Seleccione un País" itemValue="0" noSelectionOption="true"></f:selectItem>

<f:selectItems value="#{peempEmpleController.listPaises}" var="pais" itemLabel="#{pais.pepaiDescri}" itemValue="#{pais.pepaiCodigo}" />

<p:ajax event="change" listener="#{peempEmpleController.obtenerProvincias()}" update="cboProvin cboCanton cboParroq"/>

</h:selectOneMenu>

<h:outputLabel value="Cantón:\*" for="pecanCan" />

<h:selectOneMenu id="cboCanton" value="#{peempEmpleController.codCanton}" style="display: block;width:250px;margin-top: 10px;">

<f:selectItem itemLabel="Seleccione un Cantón" itemValue="0" noSelectionOption="true"></f:selectItem>

<f:selectItems value="#{peempEmpleController.listCanton}" var="cant" itemLabel="#{cant.pecanDescri}" itemValue="#{cant.pecanCantonPK.pecanCodigo}"/>

<p:ajax event="change" listener="#{peempEmpleController.obtenerParroquias()}" update="cboParroq"/>

</h:selectOneMenu>

<h:outputLabel value="Parroquia:\*" for="peparParro" />

<h:selectOneMenu id="cboParroq" value="#{peempEmpleController.codParroquia}" style="display: block;width:250px;margin-top: 10px;">

<f:selectItem itemLabel="Seleccione una Parroquia" itemValue="0" noSelectionOption="true"></f:selectItem>

<f:selectItems value="#{peempEmpleController.listParroquias}" var="parr" itemLabel="#{parr.peparDescri}" itemValue="#{parr.peparParroqPK.peparCodigo}"/>

</h:selectOneMenu>

<p:tab title="Información Laboral">

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="Salario:\*" for="peempSalario" />

<h:inputText id="peempSalario" value="#{peempEmpleController.selected.peempSalario}" title="#{bundle.CreatePeempEmpleTitle\_peempSalario}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempSalario}"/>

<h:outputLabel value="Departamento/Cargo:\*" for="coddepart" />

<h:selectOneMenu id="coddepart" value="#{peempEmpleController.selected.coddepart}" title="#{bundle.CreatePeempEmpleTitle\_coddepart}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_coddepart}">

<f:selectItems value="#{tedepDepartController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

</h:panelGrid>

</p:tab>

</p:tabView>

<p:commandButton action="#{peempEmpleController.create}" value="Guardar" ajax="false" update=":formAcceso:tablaAccesos" process="@this" icon="pi pi-check"/>

</h:panelGrid>

</p:dialog>

</h:form>

</ui:composition>

Tabla . Codificación del archivo peempEmple/ Edit.xhtml

ui:composition

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

xmlns:f="http://java.sun.com/jsf/core"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:p="http://primefaces.org/ui"

xmlns:ui="http://java.sun.com/jsf/facelets">

<h:form enctype="multipart/form-data" id="formEditarUsuario">

<p:dialog header="Editar Usuario" widgetVar="digEditarAcceso" modal="true" style="font-size: 14px" id="digEditarAcceso" width="55%" height="75vh">

<p:panelGrid columns="2">

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="Código:\*" for="peempCurp" />

<h:inputText id="peempCurp" value="#{peempEmpleController.selected.peempCurp}" title="#{bundle.CreatePeempEmpleTitle\_peempCurp}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempCurp}" readonly="true"/>

<h:outputLabel value="Cédula:\*" for="peempCedula" />

<h:inputText id="peempCedula" value="#{peempEmpleController.selected.peempCedula}" title="#{bundle.CreatePeempEmpleTitle\_peempCedula}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempCedula}" readonly="true"/>

<h:outputLabel value="Nombre:\*" for="peempNombre" />

<h:inputText id="peempNombre" value="#{peempEmpleController.selected.peempNombre}" title="#{bundle.CreatePeempEmpleTitle\_peempNombre}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempNombre}"/>

<h:outputLabel value="Apellido Paterno:\*" for="peempApellpaterno" />

<h:inputText id="peempApellpaterno" value="#{peempEmpleController.selected.peempApellpaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellpaterno}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempApellpaterno}"/>

<h:outputLabel value="Apellido Materno\*" for="peempApellmaterno" />

<h:inputText id="peempApellmaterno" value="#{peempEmpleController.selected.peempApellmaterno}" title="#{bundle.CreatePeempEmpleTitle\_peempApellmaterno}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempApellmaterno}"/>

</h:panelGrid>

<p:panelGrid columns="2" rendered="#{peempEmpleController.selected != null}" id="Image" styleClass="ui-noborder" >

<p:graphicImage name="/img/#{peempEmpleController.selected.peempFoto}" width="100px" height="100px" rendered="#{peempEmpleController.foto == null}" cache="false"/>

<p:graphicImage id="imagePreview" value="#{peempEmpleController.imagepreview}" cache="false" style="width: 100px;" rendered="#{peempEmpleController.foto != null}"/>

<p:fileUpload sizeLimit="1048500" listener="#{peempEmpleController.preview}" mode="advanced" skinSimple="true" auto="true" update="Image"/>

<p:inputText id="aux" value="#{peempEmpleController.aux}" title="aux" type="hidden"/>

<p:tabView>

<p:tab title="Información Personal">

<h:panelGrid columns="2" border="0" cellpadding="5px">

<h:outputLabel value="Sexo:\*" for="pesexCodigo" />

<h:selectOneMenu id="pesexCodigo" value="#{peempEmpleController.selected.pesexCodigo}" title="#{bundle.CreatePeempEmpleTitle\_pesexCodigo}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_pesexCodigo}">

<f:selectItems value="#{pesexSexoController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

<h:outputLabel value="Género:\*" for="codgenero" />

<h:selectOneMenu id="codgenero" value="#{peempEmpleController.selected.codgenero}" title="#{bundle.CreatePeempEmpleTitle\_codgenero}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_codgenero}">

<f:selectItems value="#{pegenGenerController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

<h:outputLabel value="Fecha de Nacimiento:\*" for="peempFecnacimiento" />

<p:calendar id="peempFecnacimiento" value="#{peempEmpleController.selected.peempFecnacimiento}" title="#{bundle.CreatePeempEmpleTitle\_peempFecnacimiento}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempFecnacimiento}"

showOn="button" navigator="true">

<f:convertDateTime pattern="dd/MM/yy" />

</p:calendar>

<p:tab title="Información Laboral">

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="Salario:\*" for="peempSalario" />

<h:inputText id="peempSalario" value="#{peempEmpleController.selected.peempSalario}" title="#{bundle.CreatePeempEmpleTitle\_peempSalario}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_peempSalario}"/>

<h:outputLabel value="Departamento:\*" for="coddepart" />

<h:selectOneMenu id="coddepart" value="#{peempEmpleController.selected.coddepart}" title="#{bundle.CreatePeempEmpleTitle\_coddepart}" required="true" requiredMessage="#{bundle.CreatePeempEmpleRequiredMessage\_coddepart}">

<f:selectItems value="#{tedepDepartController.itemsAvailableSelectOne}"/>

</h:selectOneMenu>

</h:panelGrid>

</p:tab>

</p:tabView>

<p:commandButton action="#{peempEmpleController.update}" value="Editar" ajax="false"/>

</h:panelGrid>

</p:dialog>

</h:form>

</ui:composition>

<p:tab title="Información Laboral">

<h:panelGrid columns="2" cellpadding="5px">

<h:outputLabel value="Salario:" for="peempSalario" />

<h:outputLabel id="peempSalario" value="$#{peempEmpleController.selected.peempSalario}" title="#{bundle.CreatePeempEmpleTitle\_peempSalario}" />

<h:outputLabel value="Departamento:" for="coddepart" />

<h:outputLabel id="coddepart" value="#{peempEmpleController.selected.coddepart.nombredepto}" title="#{bundle.CreatePeempEmpleTitle\_coddepart}" />

</h:panelGrid>

</p:tab>

</p:tabView>

</h:panelGrid>

</p:dialog>

</h:form>

</ui:composition>

# 4.3 CREACIÓN DE MODELOS DEL PROYECTO

### 4.3.1 CREACIÓN DE LOS ARCHIVOS DEL MODELO.

En este caso se va a generar los archivos para el modelo de forma automática a partir de la base de datos y las tablas que se tiene allí. Para ello se debe dirigir a nuevo y otros.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de un archivo para clase.

Dentro de la nueva ventana se debe seleccionar la siguiente opción.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Selección de tipo de archivo.

Una vez seleccionado se coloca en el apartado de Fuente de Base de Datos la opción “Nueva Data Source”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Elección de base de datos.

En la nueva ventana se coloca le nombre de la base de datos que se desea agregar y la conexión con dicha base.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Configuración de base de datos y conexión.

A continuación se mostrará una ventana donde se desplegarán las tablas existentes en la base, paso siguiente se debe pasar todas a la lista de tablas seleccionadas.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de tablas.

Finalmente se deberá configurar el nombre del paquete donde se crearán y se da clic en Finalizar.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Ubicación de archivos.

Se deberá haber creado una serie de archivos en el paquete ec.edu.monster.modelo similares a los siguientes.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Figura . Esquema del paquete ec.edu.monster.modelo .

### 4.3.2 CREACIÓN DEL ARCHIVO CORREO

Una de las funciones del Login es que pueda enviar correos para la recuperación y generación de contraseñas, para esto se necesita crear un nuevo archivo Java Class de la siguiente manera.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de un archivo para clase.

Dentro de la nueva ventana se debe especificar el nombre del archivo, en este caso Correo.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Nombre y ubicación de archivo.

Una vez creado el archivo se debe colocar el siguiente código.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Codificación del archivo Correo.java.

Cabe señalar que se debe activar la opción de doble factor de seguridad de la cuenta de Gmail para poder obtener un código con el cual se pueda ingresar a la cuenta y enviar el correo.

Tabla . Codificación del archivo Correo.java

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template

\*/

package ec.edu.monster.modelo;

import ec.edu.monster.controlador.EnviarCorreo;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author usuario

\*/

public class Correo {

private String usuarioCorreo;

private String contrasenia;

private String rutaArchivo;

private String nombreArchivo;

private String destino;

private String asunto;

private String mensaje;

/\*@return the usuarioCorreo

\*/

public String getUsuarioCorreo() {

return usuarioCorreo;

}

/\*\*

\* @param usuarioCorreo the usuarioCorreo to set

\*/

public void setUsuarioCorreo(String usuarioCorreo) {

this.usuarioCorreo = usuarioCorreo;

}

public String getContrasenia() {

return contrasenia;

}

/\*\*

\* @param contrasenia the contrasenia to set

\*/

public void setContrasenia(String contrasenia) {

this.contrasenia = contrasenia;

}

/\*\*

\* @return the rutaArchivo

\*/

public String getRutaArchivo() {

return rutaArchivo;

}

/\*\*

\* @param rutaArchivo the rutaArchivo to set

\*/

public void setRutaArchivo(String rutaArchivo) {

this.rutaArchivo = rutaArchivo;

}

/\*\*

\* @return the nombreArchivo

\*/

public String getNombreArchivo() {

return nombreArchivo;

}

public String getMensaje() {

return mensaje;

}

public void setMensaje(String mensaje) {

this.mensaje = mensaje;

}

public int enviarCorreo(String asunto, String mensaje, String correo)

{

Correo c = new Correo();

c.setContrasenia("pddmjldcndtgtibs");

c.setUsuarioCorreo("padillamoralesnovoagrupo6@gmail.com");

c.setAsunto(asunto);

c.setMensaje(mensaje);

c.setDestino(correo.trim());

EnviarCorreo co = new EnviarCorreo ();

if(co.enviarCorreo(c))

{

return 1;

}

else

{

return 0;

}

}

}

# 4.4 CREACIÓN DE CONTROLADORES DEL PROYECTO

Un controlador como se indicó con anterioridad es aquel que se encarga de actuar a manera de comunicador entre la vista y el modelo, recibiendo ordenes de la vista, procesándolos y modificando respectivamente el modelo, en este caso nuestro controlador recibiría ordenes desde el archivo index para verificar si las credenciales son las correctas y este solicitaría tal información a los archivos del modelo.

En este caso con la generación de los archivos del modelo y la vista ya se generaron los archivos del controlador, por lo que sólo hace falta añadir algunos archivos adicionales.

### 4.4.1 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO MENUCONTROLADOR.

De clic derecho sobre la carpeta controlador, escoja “Nuevo” y seleccione “Java Class”.

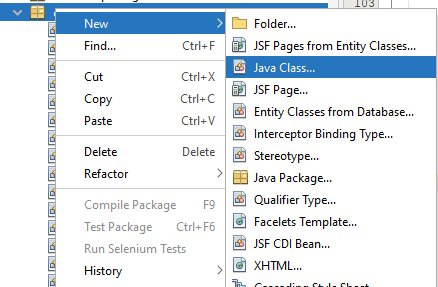


Figura . Selección de un archivo Java Class.

Le da el nombre de MenuControlador y proceda a crearlo.

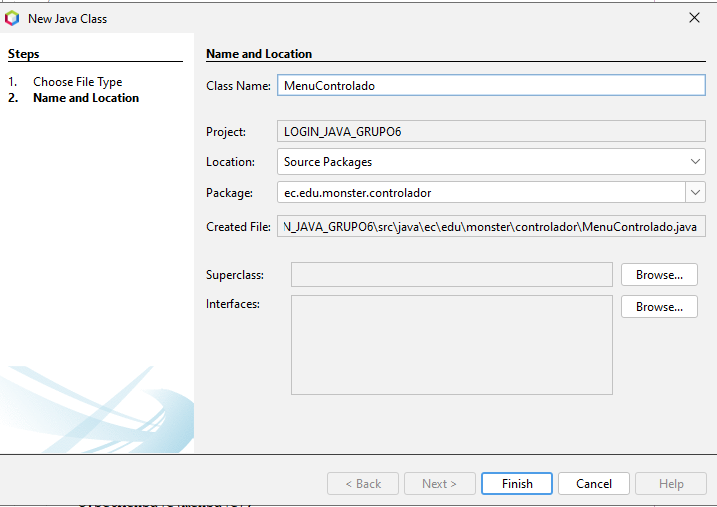


Figura . Creación del archivo MenuControlador,java

En este archivo se debe colocar el código para controlar las operaciones del menú, así como la información que se va a desplegar en el mismo, como se presenta en la siguiente tabla.

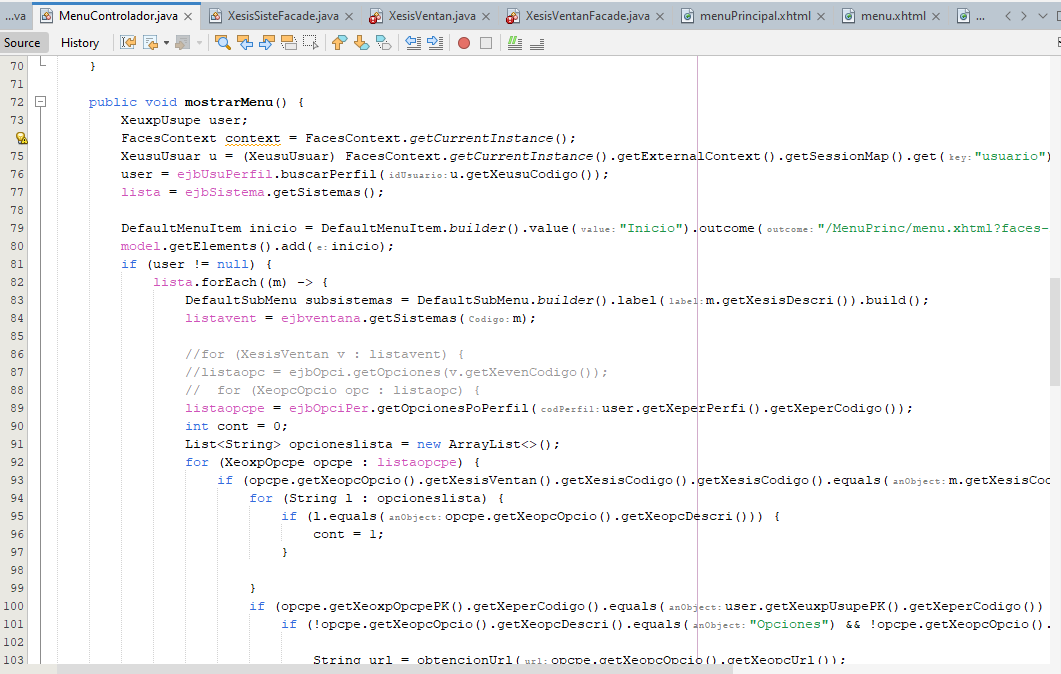


Figura . Codificación del archivo MenuControlador,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador MenuControlador.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.XeopcOpcio;

import ec.edu.monster.modelo.XeopcOpcioFacade;

import ec.edu.monster.modelo.XeoxpOpcpe;

import ec.edu.monster.modelo.XeoxpOpcpeFacade;

import ec.edu.monster.modelo.XesisSiste;

import ec.edu.monster.modelo.XeusuUsuar;

import ec.edu.monster.modelo.XeuxpUsupe;

import ec.edu.monster.modelo.XesisSisteFacade;

import ec.edu.monster.modelo.XesisVentan;

import ec.edu.monster.modelo.XesisVentanFacade;

import ec.edu.monster.modelo.XeuxpUsupeFacade;

import javax.inject.Named;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

import javax.annotation.PostConstruct;

import javax.ejb.EJB;

import javax.enterprise.context.RequestScoped;

import javax.enterprise.context.SessionScoped;

import javax.faces.context.FacesContext;

import javax.inject.Inject;

import org.primefaces.model.menu.DefaultMenuItem;

import org.primefaces.model.menu.DefaultMenuModel;

import org.primefaces.model.menu.DefaultSubMenu;

import org.primefaces.model.menu.MenuModel;

@Named(value = "menuControlador")

@SessionScoped

public class MenuControlador implements Serializable {

@Inject

private XesisSisteFacade ejbSistema;

private List<XesisSiste> lista;

private List<XeopcOpcio> listaopc;

private List<XeoxpOpcpe> listaopcpe;

private List<XesisVentan> listavent;

private MenuModel model;

private ArrayList<String> iconos;

@EJB

private XeuxpUsupeFacade ejbUsuPerfil;

@EJB

private XesisVentanFacade ejbventana;

@EJB

private XeopcOpcioFacade ejbOpci;

@EJB

private XeoxpOpcpeFacade ejbOpciPer;

/\*\*

\* Creates a new instance of MenuController

\*/

public MenuControlador() {

}

@PostConstruct

public void init() {

this.listarMenus();

model = new DefaultMenuModel();

this.mostrarMenu();

}

public void mostrarMenu() {

XeuxpUsupe user;

FacesContext context = FacesContext.getCurrentInstance();

XeusuUsuar u = (XeusuUsuar) FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("usuario");

user = ejbUsuPerfil.buscarPerfil(u.getXeusuCodigo());

lista = ejbSistema.getSistemas();

DefaultMenuItem inicio = DefaultMenuItem.builder().value("Inicio").outcome("/MenuPrinc/menu.xhtml?faces-redirect=true").build();

model.getElements().add(inicio);

if (user != null) {

lista.forEach((m) -> {

DefaultSubMenu subsistemas = DefaultSubMenu.builder().label(m.getXesisDescri()).build();

listavent = ejbventana.getSistemas(m);

listaopcpe = ejbOpciPer.getOpcionesPoPerfil(user.getXeperPerfi().getXeperCodigo());

int cont = 0;

List<String> opcioneslista = new ArrayList<>();

for (XeoxpOpcpe opcpe : listaopcpe) {

if (opcpe.getXeopcOpcio().getXesisVentan().getXesisCodigo().getXesisCodigo().equals(m.getXesisCodigo())) {

for (String l : opcioneslista) {

if (l.equals(opcpe.getXeopcOpcio().getXeopcDescri())) {

cont = 1;

}

}

if (opcpe.getXeoxpOpcpePK().getXeperCodigo().equals(user.getXeuxpUsupePK().getXeperCodigo()) && cont == 0) {

if (!opcpe.getXeopcOpcio().getXeopcDescri().equals("Opciones") && !opcpe.getXeopcOpcio().getXeopcDescri().equals("Opciones por Perfil")) {

String url = obtencionUrl(opcpe.getXeopcOpcio().getXeopcUrl());

DefaultMenuItem ventana;

if (url.startsWith("http")) {

ventana = DefaultMenuItem.builder().value(opcpe.getXeopcOpcio().getXeopcDescri()).url(url).build();

} else {

ventana = DefaultMenuItem.builder().value(opcpe.getXeopcOpcio().getXeopcDescri()).outcome(url).build();

}

System.out.println("opcpe.getXeopcOpcio().getXeopcDescri(): " + opcpe.getXeopcOpcio().getXeopcOpcioPK().getXeopcCodigo());

System.out.println("entro");

opcioneslista.add(opcpe.getXeopcOpcio().getXeopcDescri());

/\*if (opcpe.getXeopcOpcio().getXesisVentan().getXevenMensaj().equals("PROCESOS")) {

proc.getElements().

} else {

adm.getElements().add(ventana);

}\*/

subsistemas.getElements().add(ventana);

}

}

cont = 0;

}

}

// }

// }

model.getElements().add(subsistemas);

});

}

DefaultMenuItem password = DefaultMenuItem.builder().value("Cambiar Contraseña").outcome("/Password/cambiarPass.xhtml?faces-redirect=true").build();

model.getElements().add(password);

/\*DefaultMenuItem ayuda = DefaultMenuItem.builder().value("Ayuda").outcome("/Ayuda2/ayuda.xhtml?faces-redirect=true").build();

model.getElements().add(ayuda);\*/

}

public void listarMenus() {

try {

lista = ejbSistema.getSistemas();

} catch (Exception e) {

}

}

public String obtencionUrl(String url) {

String direccion;

if (!url.isEmpty()) {

if (url.startsWith("http")) {

direccion = url;

} else {

String text = url.concat(".xhtml?faces-redirect=true");

direccion = text.replace(" ", "");

}

} else {

direccion = "/Ayuda2/ayuda.xhtml?faces-redirect=true";

}

return direccion;

}

public MenuModel getModel() {

return model;

}

public void setModel(MenuModel model) {

this.model = model;

}

public void cerrarSesion() {

FacesContext.getCurrentInstance().getExternalContext().invalidateSession();

}

}

### 4.4.2 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO ENVIARCORREO.

De clic derecho sobre la carpeta controlador, escoja “Nuevo” y seleccione “Java Class”.

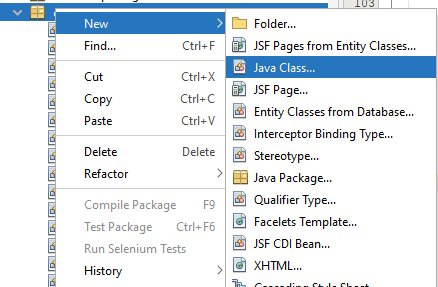


Figura . Selección de un archivo Java Class.

Le da el nombre de EnviarCorreo y proceda a crearlo.

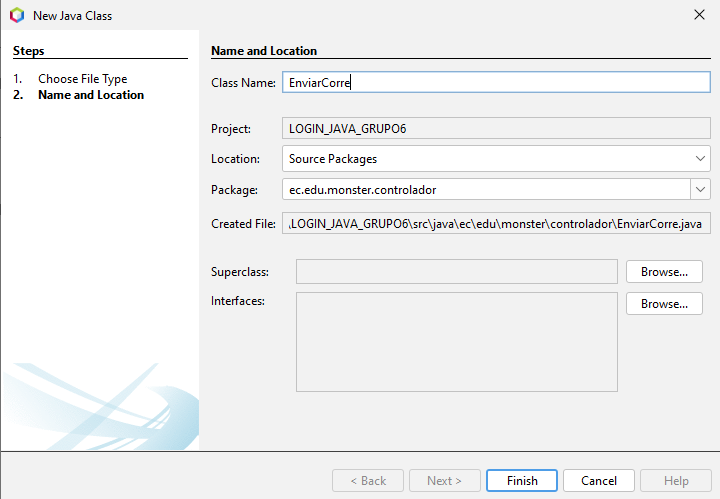


Figura . Creación del archivo EnviarCorreo,java

En este archivo se debe colocar el código para poder realizar el envío del correo con la información del modelo creado previamente.

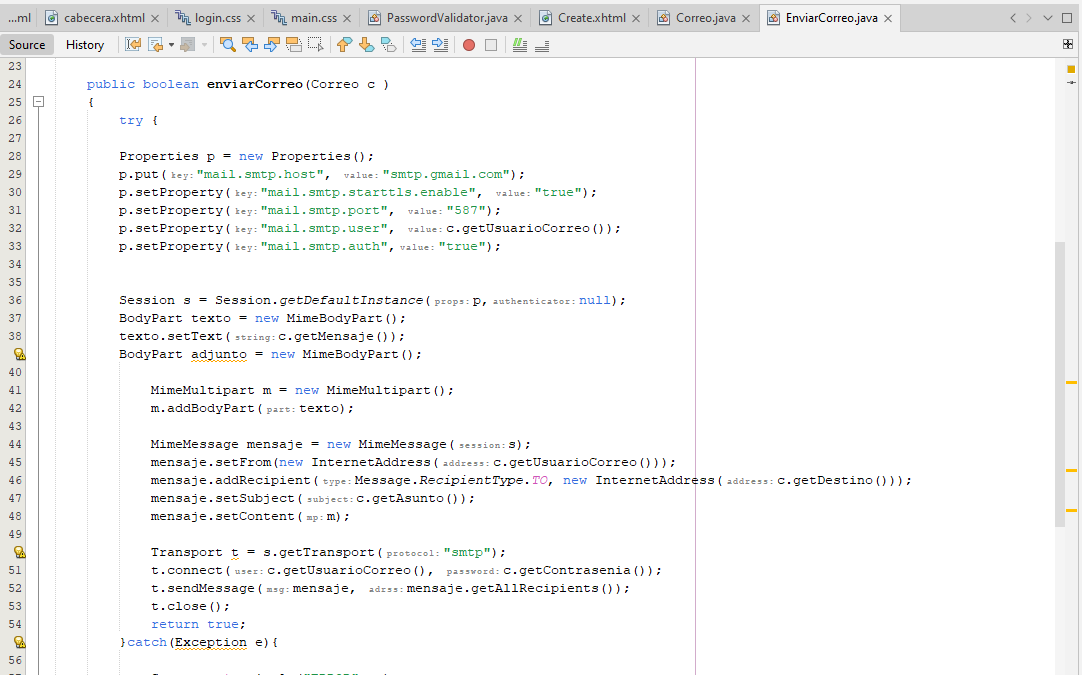


Figura . Codificación del archivo EnviarCorreo,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador EnviarCorreo.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.Correo;

import java.util.Properties;

import javax.mail.BodyPart;

import javax.mail.Message;

import javax.mail.Session;

import javax.mail.Transport;

import javax.mail.internet.InternetAddress;

import javax.mail.internet.MimeBodyPart;

import javax.mail.internet.MimeMessage;

import javax.mail.internet.MimeMultipart;

public class EnviarCorreo {

public boolean enviarCorreo(Correo c )

{

try {

Properties p = new Properties();

p.put("mail.smtp.host", "smtp.gmail.com");

p.setProperty("mail.smtp.starttls.enable", "true");

p.setProperty("mail.smtp.port", "587");

p.setProperty("mail.smtp.user", c.getUsuarioCorreo());

p.setProperty("mail.smtp.auth","true");

Session s = Session.getDefaultInstance(p,null);

BodyPart texto = new MimeBodyPart();

texto.setText(c.getMensaje());

BodyPart adjunto = new MimeBodyPart();

MimeMultipart m = new MimeMultipart();

m.addBodyPart(texto);

MimeMessage mensaje = new MimeMessage(s);

mensaje.setFrom(new InternetAddress(c.getUsuarioCorreo()));

mensaje.addRecipient(Message.RecipientType.TO, new InternetAddress(c.getDestino()));

mensaje.setSubject(c.getAsunto());

mensaje.setContent(m);

Transport t = s.getTransport("smtp");

t.connect(c.getUsuarioCorreo(), c.getContrasenia());

t.sendMessage(mensaje, mensaje.getAllRecipients());

t.close();

return true;

}catch(Exception e){

System.out.println("ERROR"+ e);

return false;

}

}

}

### 4.4.3 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO EMAILVALIDATOR.

De clic derecho sobre la carpeta controlador, escoja “Nuevo” y seleccione “Java Class”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de un archivo Java Class.

Le da el nombre de EmailValidator y proceda a crearlo.

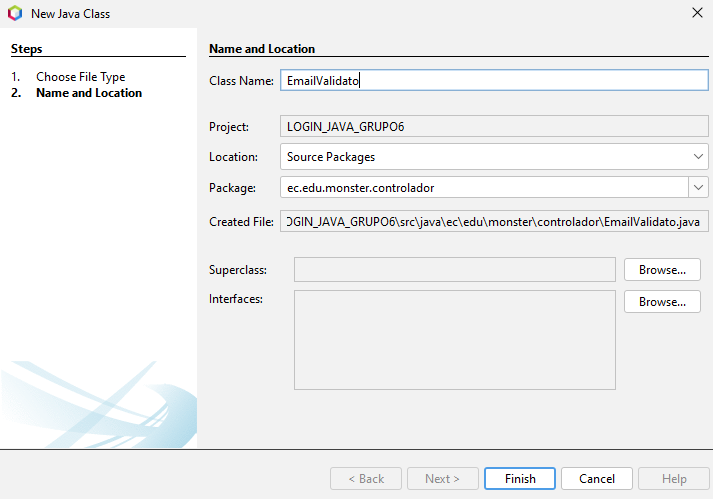


Figura . Creación del archivo EmailValidator,java

En este archivo se debe colocar el código para el control del campo email en los formularios de la vista, validando mediante Regex el input del usuario.

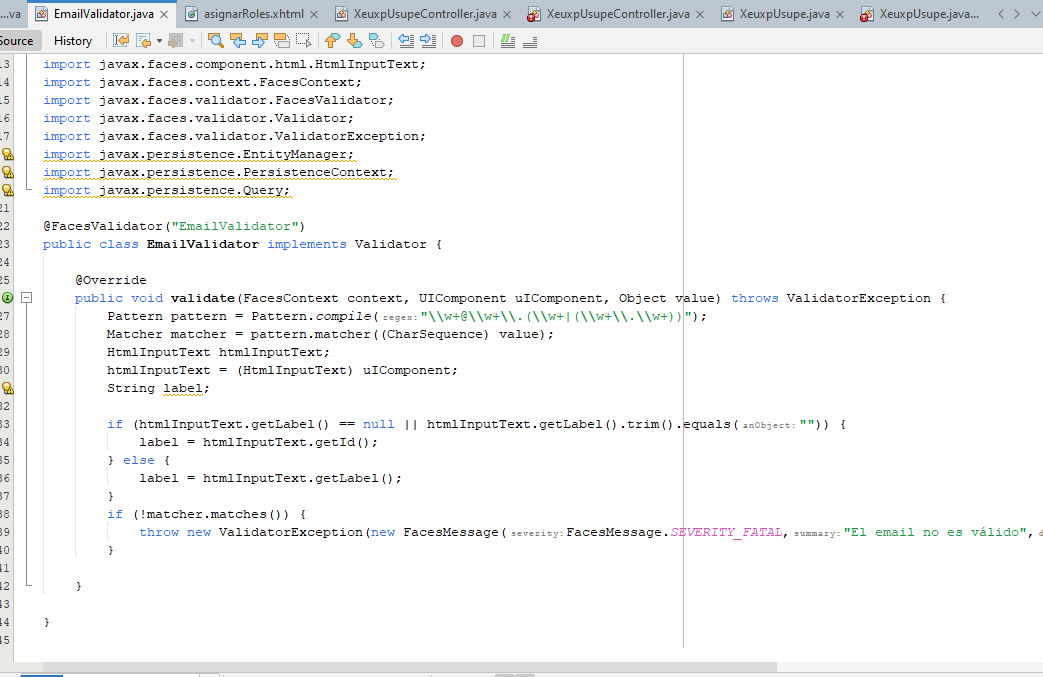


Figura . Codificación del archivo EmailValidator,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador EmailValidator.java

package ec.edu.monster.controlador;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.ejb.EJB;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.component.html.HtmlInputText;

@FacesValidator("EmailValidator")

public class EmailValidator implements Validator {

@Override

public void validate(FacesContext context, UIComponent uIComponent, Object value) throws ValidatorException {

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

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

HtmlInputText htmlInputText;

htmlInputText = (HtmlInputText) uIComponent;

String label;

if (htmlInputText.getLabel() == null || htmlInputText.getLabel().trim().equals("")) {

label = htmlInputText.getId();

} else {

label = htmlInputText.getLabel();

}

if (!matcher.matches()) {

throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY\_FATAL,"El email no es válido","\"La dirección e-mail no es válida"));

}

}

}

### 4.4.4 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO CEDULAVALIDATOR.

De clic derecho sobre la carpeta controlador, escoja “Nuevo” y seleccione “Java Class”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de un archivo Java Class.

Le da el nombre de CedulaValidator y proceda a crearlo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Creación del archivo CedulaValidator,java

En este archivo se debe colocar el código para el control del campo cédula en los formularios de la vista, validando mediante Regex el input del usuario.

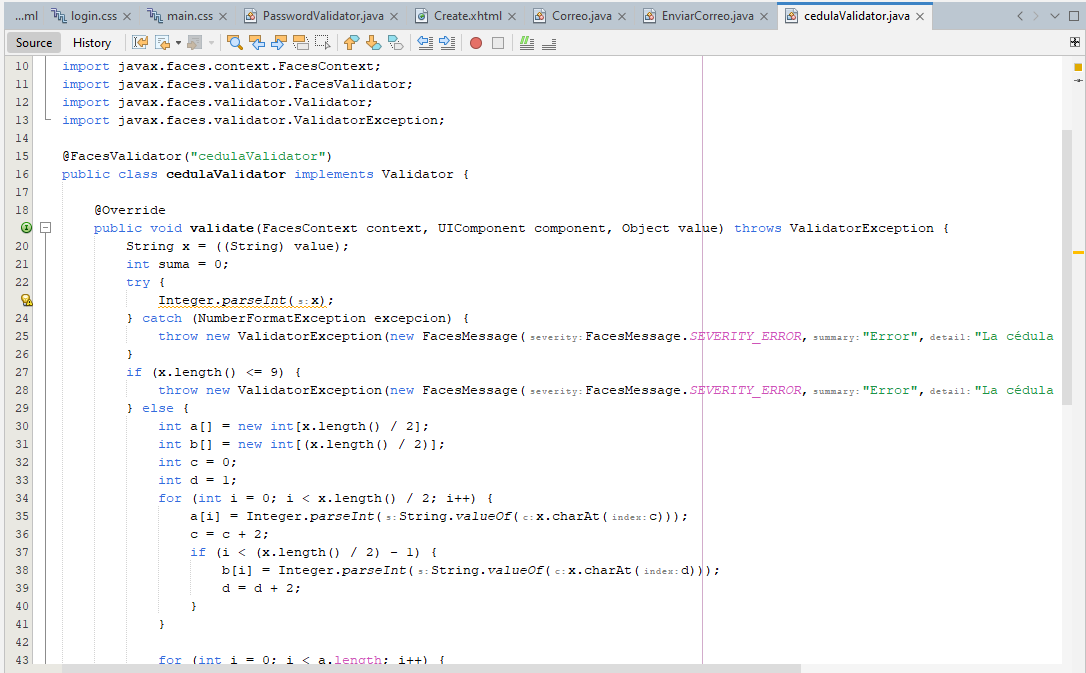


Figura . Codificación del archivo CedulaValidator,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador CedulaValidator.java

package ec.edu.monster.controlador;

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("cedulaValidator")

public class cedulaValidator implements Validator {

@Override

public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {

String x = ((String) value);

int suma = 0;

try {

Integer.parseInt(x);

} catch (NumberFormatException excepcion) {

throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY\_ERROR,"Error","La cédula debe contener solo caracteres númericos"));

}

if (x.length() <= 9) {

throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY\_ERROR,"Error","La cédula debe contener 10 dígitos"));

} else {

int a[] = new int[x.length() / 2];

int b[] = new int[(x.length() / 2)];

int c = 0;

int d = 1;

for (int i = 0; i < x.length() / 2; i++) {

a[i] = Integer.parseInt(String.valueOf(x.charAt(c)));

c = c + 2;

if (i < (x.length() / 2) - 1) {

b[i] = Integer.parseInt(String.valueOf(x.charAt(d)));

d = d + 2;

}

}

for (int i = 0; i < a.length; i++) {

a[i] = a[i] \* 2;

if (a[i] > 9) {

a[i] = a[i] - 9;

}

suma = suma + a[i] + b[i];

}

int aux = suma / 10;

int dec = (aux + 1) \* 10;

if ((dec - suma) == Integer.parseInt(String.valueOf(x.charAt(x.length() - 1)))) {

} else if (suma % 10 == 0 && x.charAt(x.length() - 1) == '0') {

} else {

throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY\_ERROR,"Error","La cédula es inválida"));

}

}

}

}

### 4.4.5 CREACIÓN Y CODIFICACIÓN DEL ARCHIVO PASSWORDVALIDATOR.

De clic derecho sobre la carpeta controlador, escoja “Nuevo” y seleccione “Java Class”.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección de un archivo Java Class.

Le da el nombre de PasswordValidator y proceda a crearlo.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Creación del archivo PasswordValidator,java

En este archivo se debe colocar el código para controlar las operaciones del menú, así como la información que se va a desplegar en el mismo, como se presenta en la siguiente tabla.

Texto

Descripción generada automáticamente

Figura . Codificación del archivo PasswordValidator,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador PasswordValidator.java

package ec.edu.monster.controlador;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.ejb.EJB;

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;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

@FacesValidator("ContraseniaValidator")

public class PasswordValidator implements Validator {

@Override

public void validate(FacesContext context, UIComponent uIComponent, Object value) throws ValidatorException {

Pattern pattern = Pattern.compile("^(?=\\S\*?[A-Z])(?=\\S\*?\\d)\\S{8,16}$");

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

HtmlInputText htmlInputText;

htmlInputText = (HtmlInputText) uIComponent;

String label;

if (htmlInputText.getLabel() == null || htmlInputText.getLabel().trim().equals("")) {

label = htmlInputText.getId();

} else {

label = htmlInputText.getLabel();

}

if (!matcher.matches()) {

throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY\_FATAL,"Contraseña inválida","\"La contraseña debe tener 8 a 16 caracteres y contener mínimo una mayúscula y un número."));

}

}

}

# 5. EJECUCIÓN DEL PROYECTO

Verifique en su compilador de Netbeans en la barra de herramientas el icono en forma triangular de color verde, será necesario presionarlo compilar el proyecto.

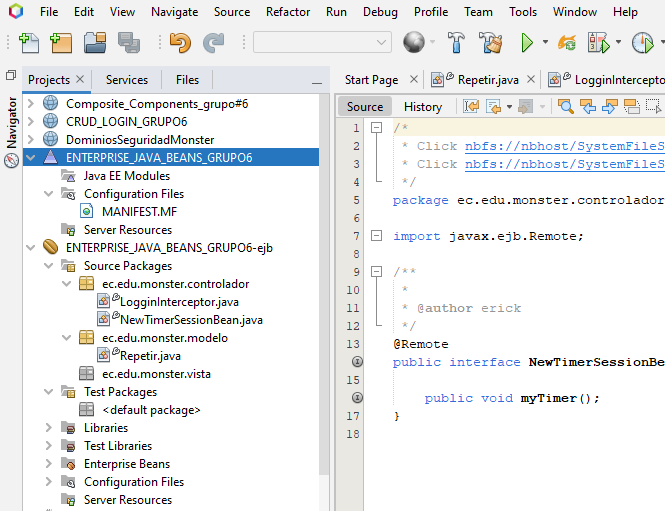


Figura . Botón para ejecutar el proyecto.

Al compilar el proyecto deberá obtener el siguiente resultado en pantalla.

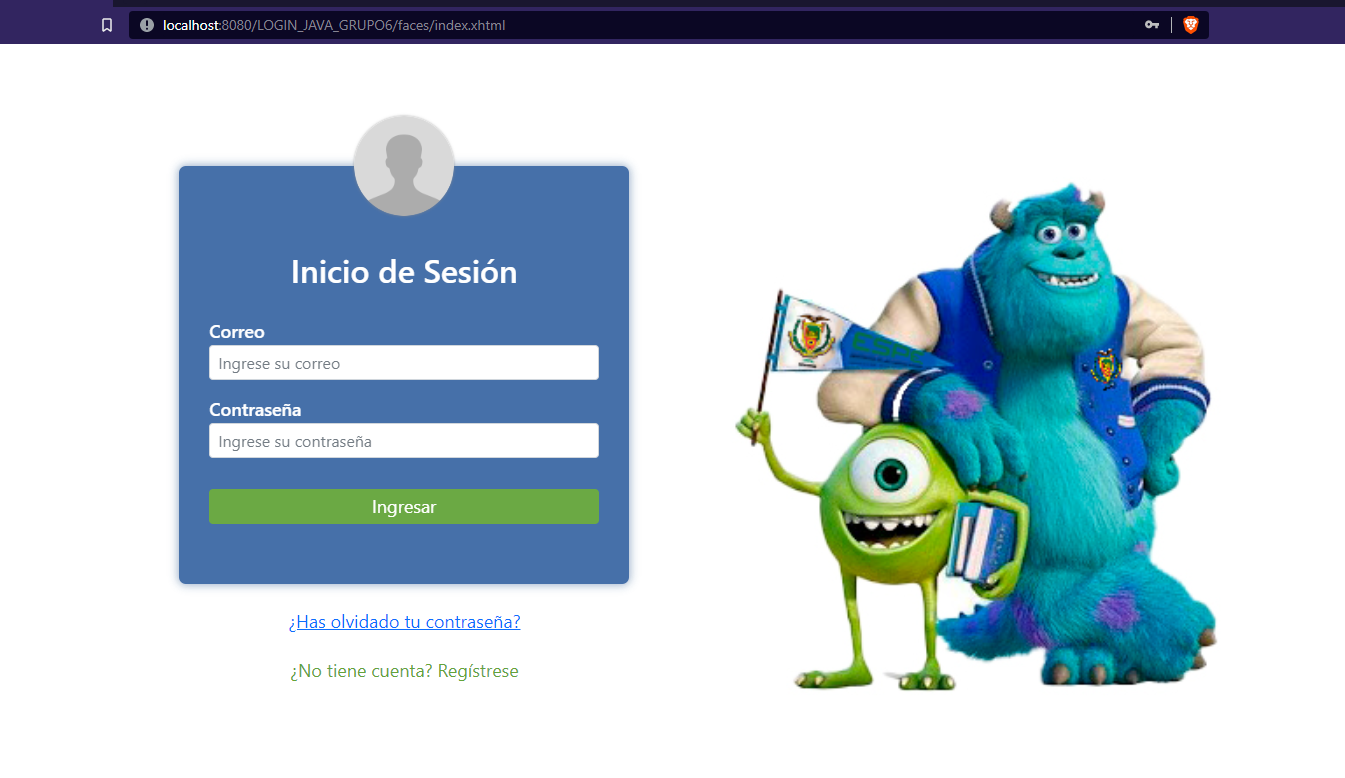


Figura . Ejecución del proyecto.

Si usted ingresa unas credenciales incorrectas aparecerá un mensaje indicando el caso.

Captura de pantalla de un celular con la imagen de una caricatura de un personaje animado

Descripción generada automáticamente con confianza baja

Figura . Mensajes de Advertencia.

Caso contrario si ingresa unas credenciales correctas, pero del perfil administrador tendrá la siguiente pantalla.

Captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente

Figura . Menú de Administrador.

Por otro lado si ingresa unas credenciales correctas, pero del perfil empleado tendrá la siguiente pantalla.

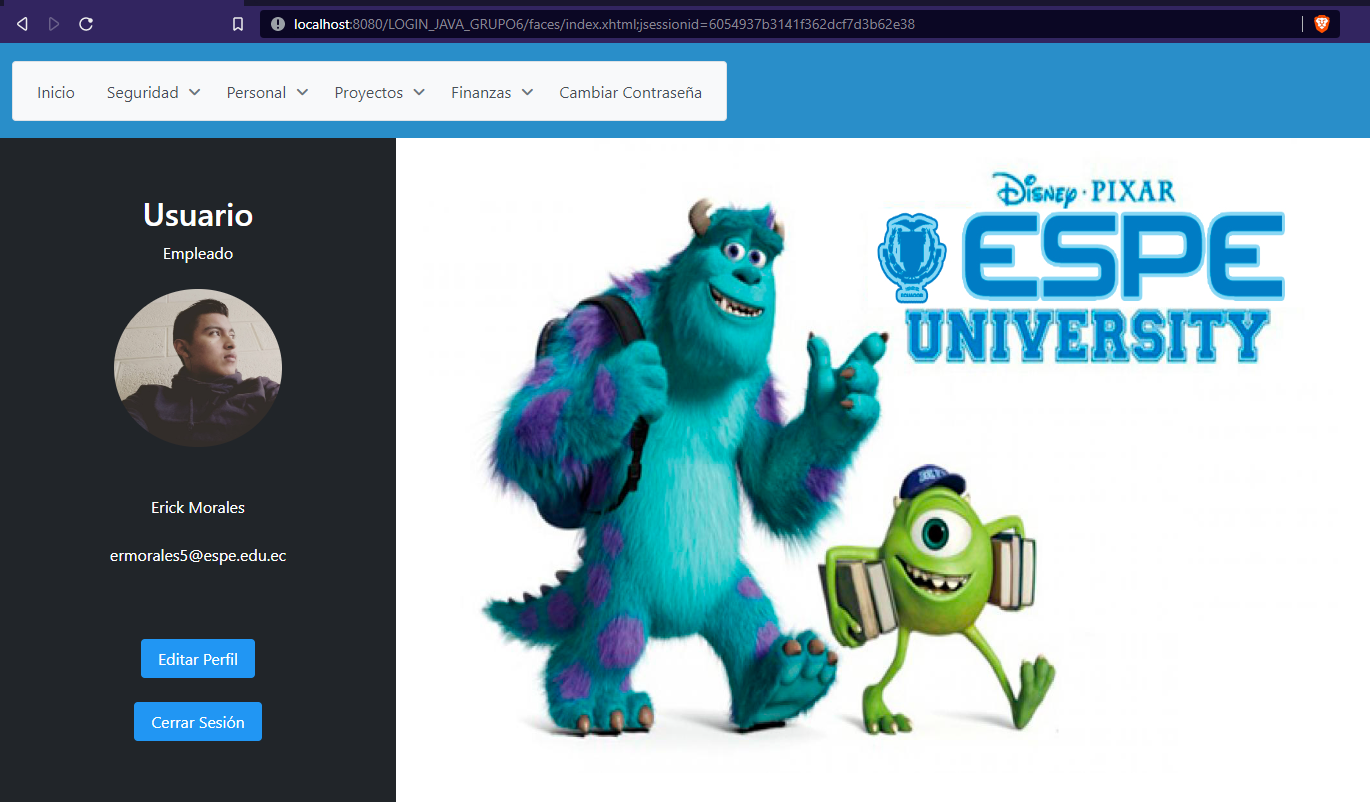


Figura . Menú de Empleado.

En caso de que quiera registrarse el formulario se verá de la siguiente manera, en este apartado se encuentran los marcos de pantalla.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura . Registro de usuario.

Los cruds se despliegan de la siguiente forma.

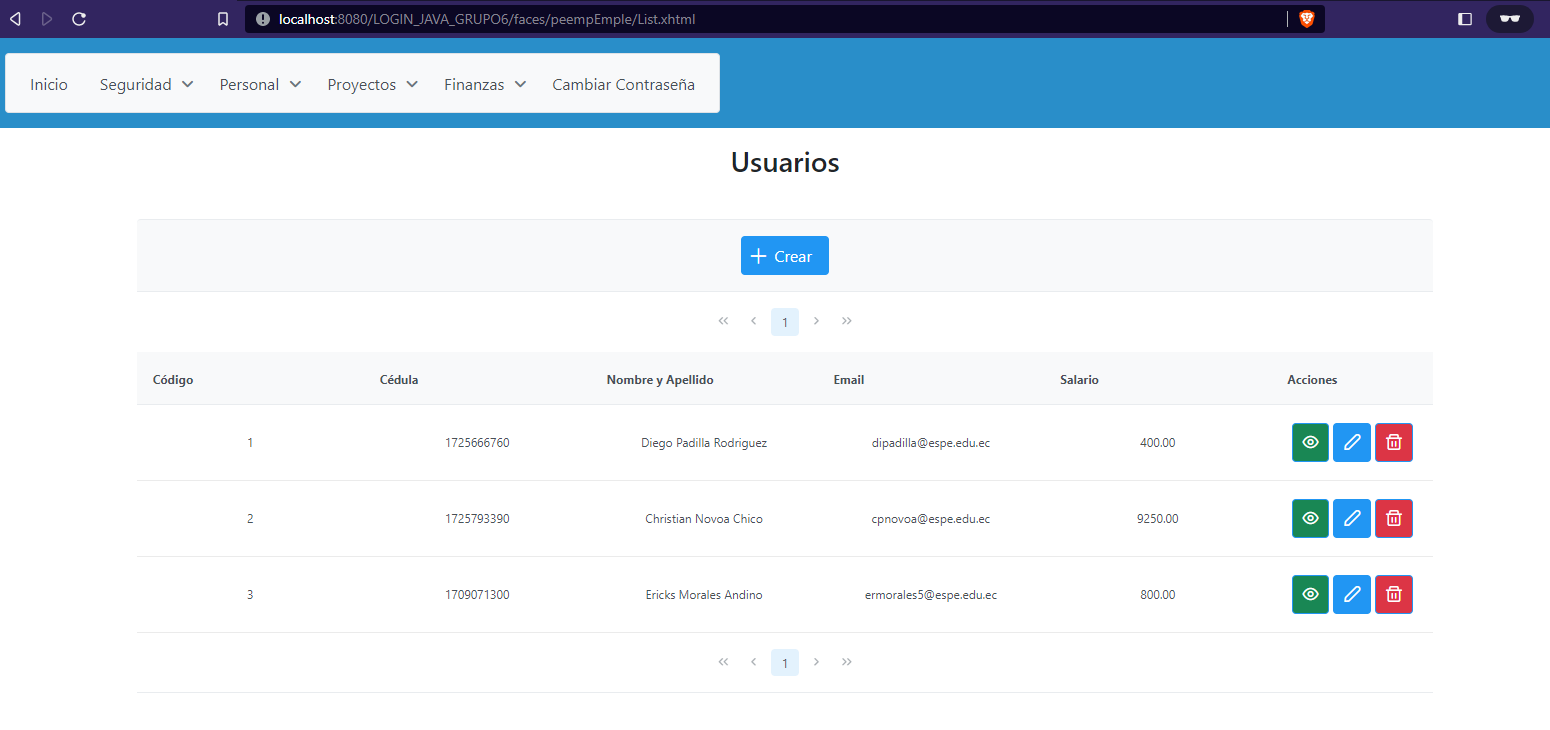


Figura . Crud de usuario.

Los dentro de los cruds se permite ver la información de cada registro en un marco de pantalla.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Marco de pantalla de información de usuario.

Al presionar en cambiar la contraseña se mostrará la siguiente pantalla.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Cambio de contraseña.

Al usuario le llega el siguiente correo que ha ingresado en el registro, en caso de ser correcto.

Imagen que contiene Forma

Descripción generada automáticamente

Figura . Correo de reseteo de contraseña.

# 6. CONCLUSIONES

* Java EE permite crear aplicaciones sumamente separadas para facilitar el mantenimiento con el uso de BEANS.
* Java EE ofrece una serie de elementos para crear programas sumamente mantenibles y fácil de programar e incluso automáticos, como lo es el uso de timers.

# 7. RECOMENDACIONES

* Se recomienda tener bien claro los conceptos de los elementos utilizados en el programa antes de desarrollar un proyecto con los mismos, puesto que se pueden utilizar métodos innecesarios o funciones que pueden ocasionar problemas a futuro.
* Se recomienda tener cuidado al momento de realizar el cambio de nombre de alguna función sobre todo en el controlador ya que un cambio en el controlador puede tener la capacidad de afectar a todo el proyecto, así como al momento de realizar la configuración de las rutas de inicio del proyecto.

# 8. BIBLIOGRAFÍA

|  |  |
| --- | --- |
| [1] | D. Web, «Desarrollo Web,» 2020 Julio 28. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: Febrero 2023]. |
| [2] | «Integración con servicios web, Java y bases de datos,» IBM, 03 04 2021. [En línea]. Available: https://www.ibm.com/docs/es/bpm/8.5.7?topic=deprecated-integrating-web-services-java-databases. [Último acceso: 23 01 2023]. |
| [3] | «Tratamiento de los datos entre las capas,» Junta de Andalucía, [En línea]. Available: https://tinyurl.com/2u8abays. [Último acceso: 23 01 2023]. |
| [4] | «CAPÍTULO 18 INTERFACES,» arkaitzgarro, [En línea]. Available: https://www.arkaitzgarro.com/java/capitulo-18.html. [Último acceso: 23 01 2023]. |
| [5] | «JavaServer Faces(JSF),» Junta de desarrollo de Andalucía, 13 11 2016. [En línea]. Available: https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/101. [Último acceso: 06 01 2023]. |
| [6] | Microsoft, «Microsoft,» [En línea]. Available: https://support.microsoft.com/es-es/office/access-sql-conceptos-b%C3%A1sicos-vocabulario-y-sintaxis-444d0303-cde1-424e-9a74-e8dc3e460671. [Último acceso: Enero 2023]. |
| [7] | Vivek, «Baeldung,» 23 Febrero 2022. [En línea]. Available: baeldung.com/jpa-entities. [Último acceso: Enero 2023]. |
| [8] | «Qué es MySQL: Características y ventajas,» openwebinars, [En línea]. Available: https://openwebinars.net/blog/que-es-mysql/. [Último acceso: 23 01 2023]. |
| [9] | J. Segovia, «todoPostgreSQL,» 7 Noviembre 2017. [En línea]. Available: https://www.todopostgresql.com/diferencias-entre-ddl-dml-y-dcl/. [Último acceso: 7 Enero 2023]. |
| [10] | J. A. Senso, «La definición de roles en la gestión de un sitio web,» Universidad de Granada, 17 11 2014. [En línea]. Available: https://tinyurl.com/yc2zbnse. [Último acceso: 23 01 2023]. |
| [11] | «MySQL Connectors,» MySQL, [En línea]. Available: https://www.mysql.com/products/connector/. [Último acceso: 03 01 2023]. |
| [12] | «Roles de base de datos,» IBM, 17 03 2021. [En línea]. Available: https://tinyurl.com/5bky97tb. [Último acceso: 23 01 2023]. |
| [13] | H. Mejía y A. Montero, «Php Progra33,» [En línea]. Available: https://phpprogra33.webcindario.com/Fundamentos%20P.O.O.html. [Último acceso: 8 Febrero 2023]. |
| [14] | «Desarrollo Web,» 8 Abril 2014. [En línea]. Available: https://desarrolloweb.com/articulos/herencia-en-programacion-orientada-objetos.html. [Último acceso: 8 Febrero 2023]. |
| [15] | IBM, «IBM,» 14 Abril 2021. [En línea]. Available: https://www.ibm.com/docs/es/i/7.3?topic=information-smtp. |
| [16] | Felipe, «HostingPlus,» 8 Marzo 2021. [En línea]. Available: https://www.hostingplus.pe/blog/protocolo-pop-que-es-y-como-funciona/. |
| [17] | J. Jimenez, «Redes Zone,» 2022 Diciembre 28. [En línea]. [Último acceso: 2023]. |
| [18] | «Cómo crear una Base de Datos con phpMyAdmin en MySQL,» disenowebakus, [En línea]. Available: https://tinyurl.com/mvys95pb. [Último acceso: 23 01 2023]. |
| [19] | O. Blancarte, «OscarBlancarte,» [En línea]. Available: https://www.oscarblancarteblog.com/tutoriales/java-persistence-api-jpa/. [Último acceso: Enero 2023]. |
| [20] | TuProgramacion, «TuProgramacion,» [En línea]. Available: http://www.tuprogramacion.com/glosario/que-es-un-orm/. [Último acceso: Enero 2022]. |