

**Tema**

USUARIOS POR PERFIL 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

USUARIOS POR PERFIL EN NETBEANS.

[1. INTRODUCCIÓN 5](#_Toc129369848)

[2. OBJETIVO 5](#_Toc129369849)

[2.1 OBJETIVO GENERAL 5](#_Toc129369850)

[2.2 OBJETIVOS ESPECÍFICOS 5](#_Toc129369851)

[3. MARCO TEÓRICO 5](#_Toc129369852)

[3.1 ARQUITECTURA EN MODELO-VISTA-CONTROLADOR 5](#_Toc129369853)

[3.2 INTEGRACION A BASES DE DATOS DESDE JAVA. 8](#_Toc129369854)

[3.3 DATA ACCESS OBJECT. 8](#_Toc129369855)

[3.4 INTERFACES JAVA. 9](#_Toc129369856)

[3.5 JAVA SERVER FACES. 10](#_Toc129369857)

[3.6 FACELETS. 10](#_Toc129369858)

[3.7 JAVA PERSISTENCE API (JPA). 10](#_Toc129369859)

[3.8 OBJETOS DE MAPEO RELACIONAL. 11](#_Toc129369860)

[3.9 ENTITY MANAGER. 13](#_Toc129369861)

[3.10 MYSQL. 14](#_Toc129369862)

[3.11 DLL Y DML 15](#_Toc129369863)

[3.12 IMPORTANCIA DE LOS ROLES EN UNA APLICACIÓN. 15](#_Toc129369864)

[3.13 MYSQL CONNECTOR Y JDBC. 16](#_Toc129369865)

[3.14 PERFILES 17](#_Toc129369866)

[3.15 FORMAS DE IMPLEMENTAR PERFILES 18](#_Toc129369867)

[3.16 RELACIÓN PERFILES Y USUARIOS 19](#_Toc129369868)

[3.17 PROGRAMACION ORIENTADA A OBJETOS (POO). 20](#_Toc129369869)

[3.18 PROTOCOLO SIMPLE DE TRANSFERENCIA DE CORREO (SMTP). 23](#_Toc129369870)

[3.19 PROTOCOLO POST OFFICE (POP). 24](#_Toc129369871)

[3.20 IMAP. 25](#_Toc129369872)

[3.21 MÉTODOS. 26](#_Toc129369873)

[3.22 TÉRMINOS. 27](#_Toc129369874)

[4. DESARROLLO 29](#_Toc129369875)

[4.1 CREACIÓN DE VISTAS DEL PROYECTO 29](#_Toc129369876)

[4.1.1 CREACIÓN DE ARCHIVO ASIGNARROLES.XHTML 29](#_Toc129369877)

[4.2 CREACIÓN DE MODELOS DEL PROYECTO 35](#_Toc129369878)

[4.2.1 CODIFICACIÓN DE ARCHIVO XEUSUUSUARFACADE.JAVA. 35](#_Toc129369879)

[4.2.2 CODIFICACIÓN DE ARCHIVO XEUXPUSUPEFACADE.JAVA. 39](#_Toc129369880)

[4.3 CREACIÓN DE CONTROLADORES DEL PROYECTO 43](#_Toc129369881)

[4.3.1 CODIFICACIÓN DEL ARCHIVO XEUSUUSUARCONTROLLER. 43](#_Toc129369882)

[4.3.2 CODIFICACIÓN DEL ARCHIVO XEUXPUSUPECONTROLLER. 58](#_Toc129369883)

[5. EJECUCIÓN DEL PROYECTO 72](#_Toc129369884)

[6. CONCLUSIONES 73](#_Toc129369885)

[7. RECOMENDACIONES 74](#_Toc129369886)

[8. BIBLIOGRAFÍA 74](#_Toc129369887)

**ÍNDICE DE IMÁGENES**

[Figura 1. Estructura de MVC 6](#_Toc129369780)

[Figura 2. Código sencillo sobre el funcionamiento de timers. 9](#_Toc129369781)

[Figura 3. Estructura JPA 11](#_Toc129369782)

[Figura 4. Interacción del ORM 12](#_Toc129369783)

[Figura 5. Estructura EntityManager 13](#_Toc129369784)

[Figura 6. Diagrama básico de sockets. 17](#_Toc129369785)

[Figura 7. Funciones de un programa dependiendo del perfil del usuario. 18](#_Toc129369786)

[Figura 8. Código sencillo sobre el funcionamiento de timers. 20](#_Toc129369787)

[Figura 9. Características de POO. 22](#_Toc129369788)

[Figura 10. Funcionamiento Correo electrónico. 23](#_Toc129369789)

[Figura 11. Último avance de CRUD realizado en personal. 29](#_Toc129369790)

[*Figura 12. Selección de un nuevo archivo.* 30](#_Toc129369791)

[*Figura 13. Creación archivo asignarRoles.xhtml* 30](#_Toc129369792)

[*Figura 14. Codificación del archivo asignarRoles.xhtml* 30](#_Toc129369793)

[Figura 15. Selección del archivo para modificar. 35](#_Toc129369794)

[Figura 16. Codificación del archivo XeusuUsuarFacade.java. 35](#_Toc129369795)

[Figura 17. Selección del archivo para modificar. 39](#_Toc129369796)

[Figura 18. Codificación del archivo XeuxpUsupeFacade.java. 39](#_Toc129369797)

[Figura 19. Selección del archivo Java Class. 43](#_Toc129369798)

[Figura 20. Codificación del archivo XeusuUsuarController,java 44](#_Toc129369799)

[Figura 21. Selección del archivo Java Class. 58](#_Toc129369800)

[Figura 22. Codificación del archivo XeuxpUsupeController,java 58](#_Toc129369801)

[Figura 23. Botón para ejecutar el proyecto. 72](#_Toc129369802)

[Figura 24. Ejecución del proyecto. 72](#_Toc129369803)

[Figura 25. Usuarios por Perfil. 73](#_Toc129369804)

[Figura 26. Asignación de un usuario a un perfil. 73](#_Toc129369805)

**ÍNDICE DE TABLAS**

[Tabla 1. Mandatos de SMTP 24](#_Toc129369772)

[Tabla 2. Métodos generalmente utilizados en conexión de java y bases de datos. 27](#_Toc129369773)

[Tabla 3. Términos 28](#_Toc129369774)

[Tabla 4. Codificación del archivo asignarRoles.xhtml 32](file:///D:\7mo%20Semestre\Distribuidas\Tercer%20Parcial\Proyecto%20Tercer%20Parcial%20Monster\05.%20DOCUMENTACION\USUARIOS%20POR%20PERFIL_NETBEANS_GRUPO6.docx#_Toc129369775)

[Tabla 5. Codificación del archivo XeusuUsuarFacade.java 37](#_Toc129369776)

[Tabla 6. Codificación del archivo XeuxpUsupeFacade.java 41](#_Toc129369777)

[Tabla 7. Codificación del archivo controlador XeusuUsuarController.java 46](file:///D:\7mo%20Semestre\Distribuidas\Tercer%20Parcial\Proyecto%20Tercer%20Parcial%20Monster\05.%20DOCUMENTACION\USUARIOS%20POR%20PERFIL_NETBEANS_GRUPO6.docx#_Toc129369778)

[Tabla 8. Codificación del archivo controlador XeuxpUsupeController.java 60](file:///D:\7mo%20Semestre\Distribuidas\Tercer%20Parcial\Proyecto%20Tercer%20Parcial%20Monster\05.%20DOCUMENTACION\USUARIOS%20POR%20PERFIL_NETBEANS_GRUPO6.docx#_Toc129369779)

# 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 asignación de perfiles a los usuarios de la empresa, que permita asignar y quitar roles de forma dinámica, sencilla e intuitiva. A través de este trabajo se pretende mejorar la eficiencia y seguridad en la gestión de perfiles de los usuarios en Screams Inc.

# OBJETIVO

### OBJETIVO GENERAL

Diseñar e implementar un sistema de gestión de perfiles de los usuarios 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.

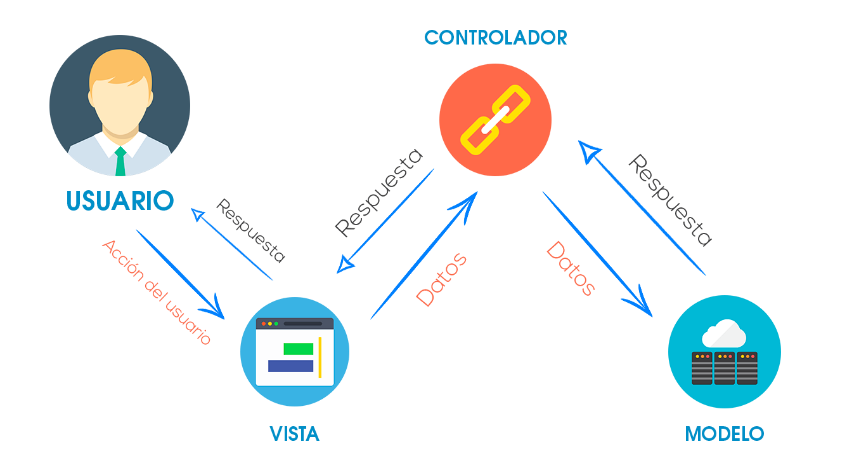


Figura 1. Estructura de MVC

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.

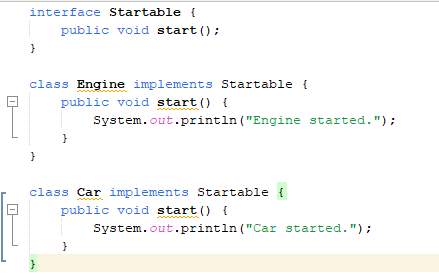


Figura 2. Código sencillo sobre el funcionamiento de timers.

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 enseñan es a utiliza el API de JDBC el cual 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.

## PERFILES

En programación, [12] los perfiles son una forma de definir un conjunto de configuraciones y características que se aplican a un usuario o grupo de usuarios en particular. En términos generales, un perfil define la forma en que un usuario interactúa con un sistema o aplicación, lo que puede incluir opciones de configuración, permisos de acceso, preferencias de interfaz, datos específicos de usuario, entre otros.

En el desarrollo de software, los perfiles pueden ser utilizados para personalizar la experiencia de los usuarios según sus necesidades y roles específicos en la aplicación. Por ejemplo, en un sistema de gestión de contenidos, se puede crear un perfil para un usuario administrador que tenga acceso a todas las funciones y herramientas, mientras que se puede crear otro perfil para un usuario normal que solo tenga acceso a ciertas funciones y herramientas.

Por lo general los perfiles permiten una mayor flexibilidad y personalización en la interacción de los usuarios con el software, al permitir que las configuraciones y características se ajusten a las necesidades y roles específicos de los usuarios.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Figura 7. Funciones de un programa dependiendo del perfil del usuario.

## FORMAS DE IMPLEMENTAR PERFILES

En el desarrollo web, existen varias formas en las que se pueden manejar los perfiles de usuarios. Algunas de las más comunes son las siguientes:

* Autenticación basada en roles: Este enfoque implica definir roles para los usuarios, como administrador, usuario regular, invitado, etc. Luego, se define el acceso a diferentes partes del sitio web en función de estos roles. Por ejemplo, el administrador puede acceder a todas las partes del sitio, mientras que los usuarios regulares solo pueden acceder a ciertas partes. La autenticación se realiza mediante un sistema de inicio de sesión, que verifica las credenciales del usuario y determina su rol.
* Autenticación basada en permisos: En este enfoque, los permisos se definen de forma granular para diferentes recursos, como páginas, funcionalidades, etc. Los usuarios pueden tener diferentes permisos para diferentes recursos. Por ejemplo, un usuario puede tener permiso para ver una página determinada, pero no para editarla. La autenticación se realiza de la misma manera que en el enfoque basado en roles.
* Perfiles de usuario personalizados: Este enfoque implica almacenar información adicional sobre los usuarios en una base de datos, como sus preferencias, historial de compras, etc. Esto permite personalizar la experiencia del usuario y ofrecer contenido y funcionalidades específicas en función de sus intereses y comportamiento.
* Autenticación externa: En este enfoque, la autenticación se realiza mediante un proveedor externo, como Google, Facebook o Twitter. Esto puede simplificar el proceso de registro e inicio de sesión para los usuarios, ya que no necesitan crear una cuenta nueva para cada sitio web. Además, puede proporcionar información adicional sobre el usuario, como su nombre y foto de perfil, que se pueden utilizar para personalizar la experiencia del usuario.

Un ejemplo sencillo de cómo se puede manejar perfiles en el desarrollo web es a través del uso de roles y permisos.

Por ejemplo, en una aplicación web de compras en línea, se pueden definir diferentes roles de usuario como "cliente", "administrador" y "vendedor". Cada rol tendría diferentes permisos y restricciones en la aplicación.

Los clientes pueden ver los productos disponibles, agregarlos a su carrito de compras y realizar pedidos, pero no pueden acceder a la información de otros usuarios o modificar productos en la plataforma.

Los administradores, por otro lado, tendrían acceso a toda la información de la plataforma, incluyendo información de usuarios y productos, y pueden agregar o eliminar productos, y realizar cambios en la configuración de la plataforma.

Los vendedores tendrían permisos para agregar nuevos productos a la plataforma y actualizar la información de los productos que han agregado, pero no tendrían acceso a la información de otros vendedores o a la información de los clientes.

De esta manera, el uso de roles y permisos permite restringir el acceso a ciertas áreas y funciones de la aplicación según el tipo de usuario, asegurando la privacidad y seguridad de los datos y mejorando la experiencia de usuario.

## RELACIÓN PERFILES Y USUARIOS

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 [13] 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 "perfiles":**

* codigo\_perfil (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.

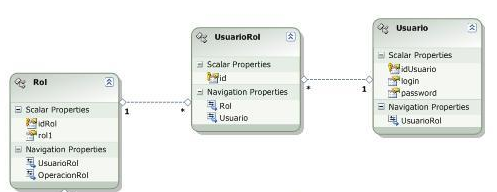


Figura 8. Código sencillo sobre el funcionamiento de timers.

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) [14] 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:** [15] 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 9. 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.

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

Es un protocolo TCP/IP [16] 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 10. 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. |

## 3.19 PROTOCOLO POST OFFICE (POP).

Es un protocolo de correo electrónico [17] 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.

## 3.20 IMAP.

IMAP (Protocolo de Mensajería de Acceso a Internet) es un protocolo de correo electrónico [18] 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 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 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.

## 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

Antes de continuar se debe tener en cuenta que el siguiente documento parte del trabajo realizado en Login\_Netbeans\_Grupo#, es decir, se omitirá la creación y codificación de algunos archivos y se asumirá su previa existencia.

En la presente practica se continuará trabajando sobre el programa realizado en el anterior proyecto, es decir, desde el Login hasta el listar usuarios en donde se presentaba el CRUD del personal con marcos de página, tal y como se puede ver en la siguiente figura.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 11. Último avance de CRUD realizado en personal.

Es por ello por lo que, a partir de ahora en el documento, se comenzará a modificar una serie de archivos que fueron creados con anterioridad, así como una serie de nuevos a ser creados.

# 4.1 CREACIÓN DE VISTAS DEL PROYECTO

4.1.1 CREACIÓN DE ARCHIVO ASIGNARROLES.XHTML

Para que el usuario pueda visualizar la asignación de roles o perfiles se debe crear una vista que englobe a los usuarios sin rol, un seleccionador con los roles disponibles y botones para poder pasar a los usuarios de un rol a otro.

Para esto primero se debe crear un nuevo archivo dando clic en la carpeta xeuxpUsupe y seleccionar Nuevo > “JSP Pages”.

Interfaz de usuario gráfica

Descripción generada automáticamente

*Figura 12. Selección de un nuevo archivo.*

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

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

*Figura 13. Creación archivo asignarRoles.xhtml*

Coloque en el mismo el siguiente código.

Texto

Descripción generada automáticamente

*Figura 14. Codificación del archivo asignarRoles.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:ui="http://xmlns.jcp.org/jsf/facelets"

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

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

<h:head>

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

<link rel="icon" href="your\_address\_icon" type="image/x-icon"></link>

</h:head>

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

<ui:define name="style">

<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"/>

</ui:define>

<ui:define name="title">

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

Tabla . Codificación del archivo asignarRoles.xhtml

</ui:define>

<ui:define name="body">

<center>

<div style="width: 75%;">

<h2>Asignación</h2>

<h:form>

<div class="content">

<div class="col s12 m7">

<h4 class="header">Usuarios x Perfil</h4>

<br/>

<div class="card-stacked">

<div id="card-header">

<p:panelGrid columns="2">

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

<p:selectOneMenu value="#{xeuxpUsupeController.perfil}" style="width: 300px">

<p:ajax event="valueChange" update="assignedUsers"/>

<f:selectItem

itemLabel="Seleccione un perfil"

itemValue=""

/>

<f:selectItems value="#{xeperPerfiController.itemsAvailableSelectMany}"

var="perfilItem"

itemLabel="#{perfilItem.xeperDescri}"

itemValue="#{perfilItem.xeperCodigo}"

/>

</p:selectOneMenu>

</p:panelGrid>

</div>

<br/>

<div class="card-content">

<p:panelGrid columns="3" style="width: 100%">

<p:panel id="users" header="Usuarios no Asignados" style="width: 100%">

<!-- <li class="collection-header"><h6>Usuarios</h6></li>-->

<p:selectOneListbox style="width: 100%;" value="#{xeuxpUsupeController.codUser}">

<f:selectItems value="#{xeusuUsuarController.usuariosNotRelated}"

var="usuario"

itemLabel="#{usuario.peempEmple.peempEmail}"

itemValue="#{usuario}"/>

</p:selectOneListbox>

</p:panel>

<p:panel header="Operaciones">

<p:panelGrid columns="1" style="width: 100%; border: 0px; ">

<p:commandButton style="width: 100%;text-align: center;" actionListener="#{xeuxpUsupeController.addOne()}" update="assignedUsers users " icon="pi pi-angle-right"/>

<p:commandButton style="width: 100%;text-align: center;" actionListener="#{xeuxpUsupeController.addAll(xeusuUsuarController.usuariosNotRelated)}" update="assignedUsers users " icon="pi pi-angle-double-right" />

<p:commandButton style="width: 100%;text-align: center;" actionListener="#{xeuxpUsupeController.deleteByCodPas()}" update="assignedUsers users " icon="pi pi-angle-left" />

<p:commandButton style="width: 100%;text-align: center;" actionListener="#{xeuxpUsupeController.deleteAll()}" update="assignedUsers users " icon="pi pi-angle-double-left" />

</p:panelGrid>

</p:panel>

<p:panel id="assignedUsers" header="Usuarios Asignados" style="width: 100%">

<p:selectOneListbox value="#{xeuxpUsupeController.codUserDel}" style="width: 100%;">

<f:selectItems value="#{xeuxpUsupeController.getUsuarios(xeuxpUsupeController.perfil)}"

var="usuario"

itemLabel="#{usuario.peempEmple.peempEmail}"

itemValue="#{usuario}"/>

</p:selectOneListbox>

</p:panel>

</p:panelGrid>

</div>

</div>

</div>

</div>

</h:form>

</div>

</center>

</ui:define>

</ui:composition>

</html>

# 4.2 CREACIÓN DE MODELOS DEL PROYECTO

### 4.2.1 CODIFICACIÓN DE ARCHIVO XEUSUUSUARFACADE.JAVA.

Como se especificó al inicio del documento, se parte de la base del proyecto anterior, es por ello que los modelos para la realización de Usuarios por Perfil ya están creados, pero se debe añadir algunos métodos nuevos a ciertos archivos.

Para ello se debe abrir la carpeta ec.edu.monster.modelo y seleccionar el archivo XeusuUsuarFacade.java.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección del archivo para modificar.

Abra el archivo y codifique los siguientes métodos dentro del mismo.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura . Codificación del archivo XeusuUsuarFacade.java.

Tabla . Codificación del archivo XeusuUsuarFacade.java

package ec.edu.monster.modelo;

import java.util.List;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

@Stateless

public class XeusuUsuarFacade extends AbstractFacade<XeusuUsuar> {

@PersistenceContext(unitName = "LOGIN\_JAVA\_GRUPO6PU")

private EntityManager em;

@Override

protected EntityManager getEntityManager() {

return em;

}

public XeusuUsuarFacade() {

super(XeusuUsuar.class);

}

public XeusuUsuar acceder(XeusuUsuar u) {

XeusuUsuar user = null;

String consulta = "";

try {

consulta = "SELECT x FROM XeusuUsuar x,PeempEmple p WHERE p.peempCurp = x.peempCurp.peempCurp AND p.peempEmail = ?1 AND x.xeusuPaswd = ?2";

Query query = em.createQuery(consulta);

query.setParameter(1, u.getPeempEmple().getPeempEmail());

query.setParameter(2, u.getXeusuPaswd());

List<XeusuUsuar> lista = query.getResultList();

if (!lista.isEmpty()) {

user = lista.get(0);

}

} catch (Exception e) {

throw e;

}

return user;

}

public XeusuUsuar accederMail(XeusuUsuar u) {

XeusuUsuar user = null;

String consulta = "";

try {

consulta = "SELECT x FROM XeusuUsuar x,PeempEmple p WHERE p.peempCurp = x.peempCurp.peempCurp AND p.peempEmail = ?1";

Query query = em.createQuery(consulta);

query.setParameter(1, u.getPeempEmple().getPeempEmail());

List<XeusuUsuar> lista = query.getResultList();

if (!lista.isEmpty()) {

user = lista.get(0);

}

} catch (Exception e) {

throw e;

}

return user;

}

user = lista.get(0);

}

} catch (Exception e) {

throw e;

}

return user;

}

public int crearCodigo() {

String consulta = "";

int maximoCodigo = 0;

try {

consulta = "SELECT MAX(x.xeusuCodigo) FROM XeusuUsuar x";

Query query = em.createQuery(consulta);

Object result = query.getSingleResult();

if (result != null) {

maximoCodigo = Integer.parseInt(result.toString())+1;

}

} catch (Exception e) {

throw e;

}

return maximoCodigo;

}

public List<XeusuUsuar> getUsuariosNotRelated() {

return getEntityManager().createNativeQuery("SELECT \* FROM xeusu\_usuar x WHERE x.XEUSU\_CODIGO NOT IN (SELECT up.XEUSU\_CODIGO from xeuxp\_usupe up);", XeusuUsuar.class)

.getResultList();

}

}

### 4.2.2 CODIFICACIÓN DE ARCHIVO XEUXPUSUPEFACADE.JAVA.

Como se especificó al inicio del documento, se parte de la base del proyecto anterior, es por ello que los modelos para la realización de Usuarios por Perfil ya están creados, pero se debe añadir algunos métodos nuevos a ciertos archivos.

Para ello se debe abrir la carpeta ec.edu.monster.modelo y seleccionar el archivo XeuxpUsupeFacade.java.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Selección del archivo para modificar.

Abra el archivo y codifique los siguientes métodos dentro del mismo.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura . Codificación del archivo XeuxpUsupeFacade.java.

Tabla . Codificación del archivo XeuxpUsupeFacade.java

package ec.edu.monster.modelo;

import ec.edu.monster.modelo.XeusuUsuar;

import ec.edu.monster.modelo.XeuxpUsupe;

import java.util.List;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

import javax.persistence.TypedQuery;

import javax.transaction.Transactional;

@Stateless

public class XeuxpUsupeFacade extends AbstractFacade<XeuxpUsupe> {

@PersistenceContext(unitName = "LOGIN\_JAVA\_GRUPO6PU")

private EntityManager em;

@Override

protected EntityManager getEntityManager() {

return em;

}

public XeuxpUsupeFacade() {

super(XeuxpUsupe.class);

}

public XeuxpUsupe buscarPerfil(String idUsuario){

XeuxpUsupe user = null;

String consulta="";

try{

consulta = "SELECT x FROM XeuxpUsupe x WHERE x.xeuxpUsupePK.xeusuCodigo = ?1";

Query query = em.createQuery(consulta);

query.setParameter(1, idUsuario);

List<XeuxpUsupe> lista = query.getResultList();

if(!lista.isEmpty()){

user=lista.get(0);

System.out.println(user.getXeusuUsuar().getXeusuPaswd());

}

}catch(Exception e){

throw e;

}

return user;

}

@Transactional

public void insertWithQuery(String codigoUsuario, String codigoPerfil, String email) {

// Date date=new Date();

java.util.Date fecha = new java.util.Date();

String[] split=codigoPerfil.split("-");

System.out.println("El código del perfil es: "+codigoPerfil);

getEntityManager().createNativeQuery("INSERT INTO xeuxp\_usupe (XEUSU\_CODIGO, XEPER\_CODIGO,XEUXP\_CODUSU,XEUXP\_FECASI,XEUXP\_FECRET) VALUES (?,?,?,?,?)").setParameter(1, codigoUsuario)

.setParameter(2,split[0]).setParameter(3, "2").setParameter(4, fecha).setParameter(5, fecha).executeUpdate();

}

public List<XeuxpUsupe> getUsuarioPerfilP(String codPerfil) {

String[] split=codPerfil.split("-");

TypedQuery<XeuxpUsupe> query

= getEntityManager().createNamedQuery("XeuxpUsupe.findByXeperCodigo", XeuxpUsupe.class);

return query.setParameter("xeperCodigo", split[0]).getResultList();

}

@Transactional

public void removeAssignation(String id\_persona) {

getEntityManager().createNativeQuery("DELETE FROM xeuxp\_usupe WHERE XEUSU\_CODIGO like ?").setParameter(1, id\_persona)

.executeUpdate();

}

public List<XeusuUsuar> findUsuarios(String codusu) {

return getEntityManager().createNativeQuery("SELECT u.\* "

+ "FROM xeuxp\_usupe up, xeusu\_usuar u "

+ "WHERE up.XEUSU\_CODIGO = u.XEUSU\_CODIGO AND up.XEPER\_CODIGO like ? ", XeusuUsuar.class)

.setParameter(1, codusu).getResultList();

}

}

# 4.3 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.

### 4.3.1 CODIFICACIÓN DEL ARCHIVO XEUSUUSUARCONTROLLER.

De clic izquierdo sobre la carpeta ec.edu.monster.controlador y escoja el archivo bajo el nombre XeusuUsuarController.java

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura . Selección del archivo Java Class.

Se procede a colocar la siguiente codificación para utilizar los métodos del modelo y mandarlos a las vistas para organizar la información con marcos de página.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura . Codificación del archivo XeusuUsuarController,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador XeusuUsuarController.java

package ec.edu.monster.controlador;

import com.google.gson.JsonObject;

import ec.edu.monster.modelo.XeusuUsuar;

import ec.edu.monster.controlador.util.JsfUtil;

import ec.edu.monster.controlador.util.JsfUtil.PersistAction;

import ec.edu.monster.controlador.util.PaginationHelper;

import ec.edu.monster.modelo.Correo;

import ec.edu.monster.modelo.XeusuUsuarFacade;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.Serializable;

import java.net.URL;

import java.util.List;

import java.util.ResourceBundle;

import javax.ejb.EJB;

import javax.inject.Named;

import javax.enterprise.context.SessionScoped;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.FacesConverter;

import javax.faces.model.DataModel;

import javax.faces.model.ListDataModel;

import javax.faces.model.SelectItem;

import com.google.gson.JsonParser;

import org.apache.commons.codec.digest.DigestUtils;

@Named("xeusuUsuarController")

@SessionScoped

public class XeusuUsuarController implements Serializable {

private XeusuUsuar current;

private DataModel items = null;

private String newPassword;

private String repPassword;

@EJB

private ec.edu.monster.modelo.XeusuUsuarFacade ejbFacade;

private PaginationHelper pagination;

private int selectedItemIndex;

public String getRepPassword() {

return repPassword;

}

public void setRepPassword(String repPassword) {

this.repPassword = repPassword;

}

public String getNewPassword() {

return newPassword;

}

public void setNewPassword(String newPassword) {

this.newPassword = newPassword;

}

public XeusuUsuarController() {

}

public XeusuUsuar getSelected() {

if (current == null) {

current = new XeusuUsuar();

selectedItemIndex = -1;

}

return current;

}

private XeusuUsuarFacade getFacade() {

return ejbFacade;

}

public PaginationHelper getPagination() {

if (pagination == null) {

pagination = new PaginationHelper(10) {

@Override

public int getItemsCount() {

return getFacade().count();

}

@Override

public DataModel createPageDataModel() {

return new ListDataModel(getFacade().findRange(new int[]{getPageFirstItem(), getPageFirstItem() + getPageSize()}));

}

};

}

return pagination;

}

public String prepareList() {

recreateModel();

return "List";

}

public String prepareView() {

current = (XeusuUsuar) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

return "View";

}

public String prepareCreate() {

current = new XeusuUsuar();

selectedItemIndex = -1;

return "Create";

}

public String create() {

try {

getFacade().create(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeusuUsuarCreated"));

return prepareCreate();

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

return null;

}

}

public String prepareEdit() {

current = (XeusuUsuar) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

return "Edit";

}

public String update() {

try {

getFacade().edit(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeusuUsuarUpdated"));

return "View";

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

return null;

}

}

public String destroy() {

current = (XeusuUsuar) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

performDestroy();

recreatePagination();

recreateModel();

return "List";

}

public String destroyAndView() {

performDestroy();

recreateModel();

updateCurrentItem();

if (selectedItemIndex >= 0) {

return "View";

} else {

// all items were removed - go back to list

recreateModel();

return "List";

}

}

private void performDestroy() {

try {

getFacade().remove(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeusuUsuarDeleted"));

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

}

}

private void updateCurrentItem() {

int count = getFacade().count();

if (selectedItemIndex >= count) {

// selected index cannot be bigger than number of items:

selectedItemIndex = count - 1;

// go to previous page if last page disappeared:

if (pagination.getPageFirstItem() >= count) {

pagination.previousPage();

}

}

if (selectedItemIndex >= 0) {

current = getFacade().findRange(new int[]{selectedItemIndex, selectedItemIndex + 1}).get(0);

}

}

public DataModel getItems() {

if (items == null) {

items = getPagination().createPageDataModel();

}

return items;

}

private void recreateModel() {

items = null;

}

private void recreatePagination() {

pagination = null;

}

public String next() {

getPagination().nextPage();

recreateModel();

return "List";

}

public String previous() {

getPagination().previousPage();

recreateModel();

return "List";

}

public SelectItem[] getItemsAvailableSelectMany() {

return JsfUtil.getSelectItems(ejbFacade.findAll(), false);

}

public List<PeempEmple> getEmpleadoList() {

if(empleadoList==null)

empleadoList=ejbFacade.obtenerEmpleados();

return empleadoList;

}

public PeempEmple getSelected() {

if (current == null) {

current = new PeempEmple();

selectedItemIndex = -1;

}

return current;

}

public void setSelected(PeempEmple selected) {

this.selected = selected;

}

public String getAux() {

return aux;

}

public void setAux(String aux) {

this.aux = aux;

}

public List<PepaiPais> getListPaises() {

if(listPaises==null)

listPaises=ejbPais.findAll();

return listPaises;

}

public void setListPaises(List<PepaiPais> listPaises) {

this.listPaises = listPaises;

}

} else {

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_WARN, "Email o Contraseña Incorrecta!!", "Email o Password Incorrectos!!"));

}

} else {

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_WARN, "Captcha Inválido!", "Valide correctamente el captcha para iniciar sesión"));

}

} catch (Exception e) {

FacesContext.getCurrentInstance().addMessage("formLogin", new FacesMessage(FacesMessage.SEVERITY\_FATAL, "Aviso" + e.toString(), "Error"));

}

return ruta;

}

public boolean validarCaptcha() {

String captchaResponse = FacesContext.getCurrentInstance()

.getExternalContext().getRequestParameterMap().get("g-recaptcha-response");

boolean valid = false;

try {

String secretKey = "6LcwYHokAAAAADy1zQ0hkTzwROB2iV8ZnGLbF7hx";

String url = "https://www.google.com/recaptcha/api/siteverify?secret=" + secretKey + "&response=" + captchaResponse;

String response = readUrlContent(url);

JsonParser jsonParser = new JsonParser();

JsonObject jsonObject = jsonParser.parse(response).getAsJsonObject();

valid = jsonObject.get("success").getAsBoolean();

} catch (Exception e) {

e.printStackTrace();

}

return valid;

}

private String readUrlContent(String urlString) throws IOException {

URL url = new URL(urlString);

BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()));

StringBuilder sb = new StringBuilder();

String line;

while ((line = reader.readLine()) != null) {

sb.append(line);

}

reader.close();

return sb.toString();

}

public void cerrarSesion() {

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

}

public String obtenerNameFoto() {

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

if (u.getPeempEmple().getPeempFoto().equals(null) || u.getPeempEmple().getPeempFoto().equals("")) {

return "user.png";

} else {

return u.getPeempEmple().getPeempFoto();

}

}

public String cambiarPassword() {

String ruta = "";

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

try {

if (newPassword.equals(repPassword)) {

String password = DigestUtils.md5Hex(newPassword);

u.setXeusuPiefir("1");

//u.setXeusuPaswd(newPassword);

u.setXeusuPaswd(password);

getFacade().edit(u);

ruta = "index";

} else {

System.out.println("llego");

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_WARN, "Aviso", "Las contraseñas deben ser iguales"));

}

} catch (Exception e) {

throw e;

}

return ruta;

}

public String resetearPassword() {

String ruta = "";

int resp;

XeusuUsuar u;

try {

u = this.ejbFacade.accederMail(this.current);

if (u != null) {

String passsword = this.current.getPeempEmple().generarContraseña();

String encript = DigestUtils.md5Hex(passsword);

Correo c = new Correo();

resp = c.enviarCorreo("Reseteo de Contraseña", "Su contraseña reseteada es: " + passsword + "\n\nInicie sesión de con esta contraseña para poder cambiarla.", this.current.getPeempEmple().getPeempEmail());

ruta = "index";//?faces-redirect=true

u.setXeusuPaswd(encript);

u.setXeusuPiefir("0");

getFacade().edit(u);

if (resp == 1) {

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_INFO, "Éxito", "Contraseña restablecida correctamente, revise su correo"));

}

} else {

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_WARN, "Aviso", "Email No Registrado!!"));

}

} catch (Exception e) {

FacesContext.getCurrentInstance().addMessage(null, new FacesMessage(FacesMessage.SEVERITY\_FATAL, "Aviso", "Error"));

}

return ruta;

}

public List<XeusuUsuar> getUsuariosNotRelated() {

System.out.println(getFacade().getUsuariosNotRelated());

return getFacade().getUsuariosNotRelated();

}

@FacesConverter(forClass = XeusuUsuar.class)

public static class XeusuUsuarControllerConverter implements Converter {

@Override

public Object getAsObject(FacesContext facesContext, UIComponent component, String value) {

if (value == null || value.length() == 0) {

return null;

}

XeusuUsuarController controller = (XeusuUsuarController) facesContext.getApplication().getELResolver().

getValue(facesContext.getELContext(), null, "xeusuUsuarController");

return controller.getXeusuUsuar(getKey(value));

}

java.lang.String getKey(String value) {

java.lang.String key;

key = value;

return key;

}

String getStringKey(java.lang.String value) {

StringBuilder sb = new StringBuilder();

sb.append(value);

return sb.toString();

}

@Override

public String getAsString(FacesContext facesContext, UIComponent component, Object object) {

if (object == null) {

return null;

}

if (object instanceof XeusuUsuar) {

XeusuUsuar o = (XeusuUsuar) object;

return getStringKey(o.getXeusuCodigo());

} else {

throw new IllegalArgumentException("object " + object + " is of type " + object.getClass().getName() + "; expected type: " + XeusuUsuar.class.getName());

}

}

}

### 4.3.2 CODIFICACIÓN DEL ARCHIVO XEUXPUSUPECONTROLLER.

De clic izquierdo sobre la carpeta ec.edu.monster.controlador y escoja el archivo bajo el nombre XeuxpUsupeController.java

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura . Selección del archivo Java Class.

Se procede a colocar la siguiente codificación para utilizar los métodos del modelo y mandarlos a las vistas para organizar la información con marcos de página.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura . Codificación del archivo XeuxpUsupeController,java

Tabla 3. Codificación del archivo index.jsp

Tabla . Codificación del archivo controlador XeuxpUsupeController.java

package ec.edu.monster.controlador;

import ec.edu.monster.modelo.XeuxpUsupe;

import ec.edu.monster.controlador.util.JsfUtil;

import ec.edu.monster.controlador.util.JsfUtil.PersistAction;

import ec.edu.monster.controlador.util.PaginationHelper;

import ec.edu.monster.modelo.XeusuUsuar;

import ec.edu.monster.modelo.XeuxpUsupeFacade;

import java.io.Serializable;

import java.util.List;

import java.util.ResourceBundle;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.ejb.EJB;

import javax.ejb.EJBException;

import javax.inject.Named;

import javax.enterprise.context.SessionScoped;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.FacesConverter;

import javax.faces.model.DataModel;

import javax.faces.model.ListDataModel;

@Named("xeuxpUsupeController")

@SessionScoped

public class XeuxpUsupeController implements Serializable {

private XeuxpUsupe current;

private DataModel items = null;

@EJB

private ec.edu.monster.modelo.XeuxpUsupeFacade ejbFacade;

private XeuxpUsupe selected;

private String perfil = "";

private XeusuUsuar codUser = new XeusuUsuar();

private XeusuUsuar codUserDel = new XeusuUsuar();

private PaginationHelper pagination;

private int selectedItemIndex;

public XeuxpUsupeController() {

}

public XeuxpUsupe getSelected() {

return selected;

}

public void setSelected(XeuxpUsupe selected) {

this.selected = selected;

}

protected void setEmbeddableKeys() {

selected.getXeuxpUsupePK().setXeperCodigo(selected.getXeperPerfi().getXeperCodigo());

selected.getXeuxpUsupePK().setXeusuCodigo(selected.getXeusuUsuar().getXeusuCodigo());

}

protected void initializeEmbeddableKey() {

selected.setXeuxpUsupePK(new ec.edu.monster.modelo.XeuxpUsupePK());

}

private XeuxpUsupeFacade getFacade() {

return ejbFacade;

}

public PaginationHelper getPagination() {

if (pagination == null) {

pagination = new PaginationHelper(10) {

@Override

public int getItemsCount() {

return getFacade().count();

}

@Override

public DataModel createPageDataModel() {

return new ListDataModel(getFacade().findRange(new int[]{getPageFirstItem(), getPageFirstItem() + getPageSize()}));

}

};

}

return pagination;

}

public String prepareList() {

recreateModel();

return "List";

}

public String prepareView() {

current = (XeuxpUsupe) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

return "View";

}

public String prepareCreate() {

current = new XeuxpUsupe();

current.setXeuxpUsupePK(new ec.edu.monster.modelo.XeuxpUsupePK());

selectedItemIndex = -1;

return "Create";

}

public String create() {

try {

current.getXeuxpUsupePK().setXeusuCodigo(current.getXeusuUsuar().getXeusuCodigo());

current.getXeuxpUsupePK().setXeperCodigo(current.getXeperPerfi().getXeperCodigo());

getFacade().create(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeuxpUsupeCreated"));

return prepareCreate();

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

return null;

}

}

public String prepareEdit() {

current = (XeuxpUsupe) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

return "Edit";

}

public void addOne() {

// System.out.println(usuario);

getFacade().insertWithQuery(codUser.getXeusuCodigo(),perfil, codUser.getPeempEmple().getPeempEmail());

}

public void addAll(List<XeusuUsuar> users) {

for (int i = 0; i < users.size(); i++) {

getFacade().insertWithQuery(users.get(i).getXeusuCodigo(),perfil,users.get(i).getPeempEmple().getPeempEmail());

}

}

public void deleteAll() {

List<XeuxpUsupe> users = getFacade().getUsuarioPerfilP(perfil);

for (int i = 0; i < users.size(); i++) {

getFacade().remove(users.get(i));

}

}

public void deleteByCodPas() {

getFacade().removeAssignation(codUserDel.getXeusuCodigo());

}

public List<XeusuUsuar> getUsuarios(String id) {

// System.out.println( "usuario" +getFacade().findUsuarios(id).get(0).getXeusuUsuarPK().getXeusuEmail());

String[] splitId=id.split("-");

if ((getFacade().findUsuarios(splitId[0]).size()>0)) {

getFacade().findUsuarios(splitId[0]).get(0).getPeempEmple().getPeempEmail();

}

return getFacade().findUsuarios(splitId[0]);

}

public String update() {

try {

current.getXeuxpUsupePK().setXeusuCodigo(current.getXeusuUsuar().getXeusuCodigo());

current.getXeuxpUsupePK().setXeperCodigo(current.getXeperPerfi().getXeperCodigo());

getFacade().edit(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeuxpUsupeUpdated"));

return "View";

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

return null;

}

}

public String destroy() {

current = (XeuxpUsupe) getItems().getRowData();

selectedItemIndex = pagination.getPageFirstItem() + getItems().getRowIndex();

performDestroy();

recreatePagination();

recreateModel();

return "List";

}

public String destroyAndView() {

performDestroy();

recreateModel();

updateCurrentItem();

if (selectedItemIndex >= 0) {

return "View";

} else {

// all items were removed - go back to list

recreateModel();

return "List";

}

}

private void performDestroy() {

try {

getFacade().remove(current);

JsfUtil.addSuccessMessage(ResourceBundle.getBundle("/Bundle").getString("XeuxpUsupeDeleted"));

} catch (Exception e) {

JsfUtil.addErrorMessage(e, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

}

}

private void updateCurrentItem() {

int count = getFacade().count();

if (selectedItemIndex >= count) {

// selected index cannot be bigger than number of items:

selectedItemIndex = count - 1;

// go to previous page if last page disappeared:

if (pagination.getPageFirstItem() >= count) {

pagination.previousPage();

}

}

if (selectedItemIndex >= 0) {

current = getFacade().findRange(new int[]{selectedItemIndex, selectedItemIndex + 1}).get(0);

}

}

public DataModel getItems() {

if (items == null) {

items = getPagination().createPageDataModel();

}

return items;

}

private void recreateModel() {

items = null;

}

private void recreatePagination() {

pagination = null;

}

public String next() {

getPagination().nextPage();

recreateModel();

return "List";

}

public String previous() {

getPagination().previousPage();

recreateModel();

return "List";

}

private void persist(PersistAction persistAction, String successMessage) {

if (selected != null) {

setEmbeddableKeys();

try {

if (persistAction != PersistAction.DELETE) {

getFacade().edit(selected);

} else {

getFacade().remove(selected);

}

JsfUtil.addSuccessMessage(successMessage);

} catch (EJBException ex) {

String msg = "";

Throwable cause = ex.getCause();

if (cause != null) {

msg = cause.getLocalizedMessage();

}

if (msg.length() > 0) {

JsfUtil.addErrorMessage(msg);

} else {

JsfUtil.addErrorMessage(ex, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

}

} catch (Exception ex) {

Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);

JsfUtil.addErrorMessage(ex, ResourceBundle.getBundle("/Bundle").getString("PersistenceErrorOccured"));

}

}

}

public XeuxpUsupe getXeuxpUsupe(ec.edu.monster.modelo.XeuxpUsupePK id) {

return getFacade().find(id);

}

public List<XeuxpUsupe> getItemsAvailableSelectMany() {

return getFacade().findAll();

}

public List<XeuxpUsupe> getItemsAvailableSelectOne() {

return getFacade().findAll();

}

public String getPerfil() {

return perfil;

}

public void setPerfil(String perfil) {

this.perfil = perfil;

}

public XeusuUsuar getCodUser() {

return codUser;

}

public void setCodUser(XeusuUsuar codUser) {

this.codUser = codUser;

}

public XeusuUsuar getCodUserDel() {

return codUserDel;

}

public void setCodUserDel(XeusuUsuar codUserDel) {

this.codUserDel = codUserDel;

}

@FacesConverter(forClass = XeuxpUsupe.class)

public static class XeuxpUsupeControllerConverter implements Converter {

private static final String SEPARATOR = "#";

private static final String SEPARATOR\_ESCAPED = "\\#";

@Override

public Object getAsObject(FacesContext facesContext, UIComponent component, String value) {

if (value == null || value.length() == 0) {

return null;

}

XeuxpUsupeController controller = (XeuxpUsupeController) facesContext.getApplication().getELResolver().

getValue(facesContext.getELContext(), null, "xeuxpUsupeController");

return controller.getXeuxpUsupe(getKey(value));

}

ec.edu.monster.modelo.XeuxpUsupePK getKey(String value) {

ec.edu.monster.modelo.XeuxpUsupePK key;

String values[] = value.split(SEPARATOR\_ESCAPED);

key = new ec.edu.monster.modelo.XeuxpUsupePK();

key.setXeusuCodigo(values[0]);

key.setXeperCodigo(values[1]);

key.setXeuxpCodusu(values[2]);

return key;

}

String getStringKey(ec.edu.monster.modelo.XeuxpUsupePK value) {

StringBuilder sb = new StringBuilder();

sb.append(value.getXeusuCodigo());

sb.append(SEPARATOR);

sb.append(value.getXeperCodigo());

sb.append(SEPARATOR);

sb.append(value.getXeuxpCodusu());

return sb.toString();

}

@Override

public String getAsString(FacesContext facesContext, UIComponent component, Object object) {

if (object == null) {

return null;

}

if (object instanceof XeuxpUsupe) {

XeuxpUsupe o = (XeuxpUsupe) object;

return getStringKey(o.getXeuxpUsupePK());

} else {

Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, "object {0} is of type {1}; expected type: {2}", new Object[]{object, object.getClass().getName(), XeuxpUsupe.class.getName()});

return null;

}

}

}

}

# 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.

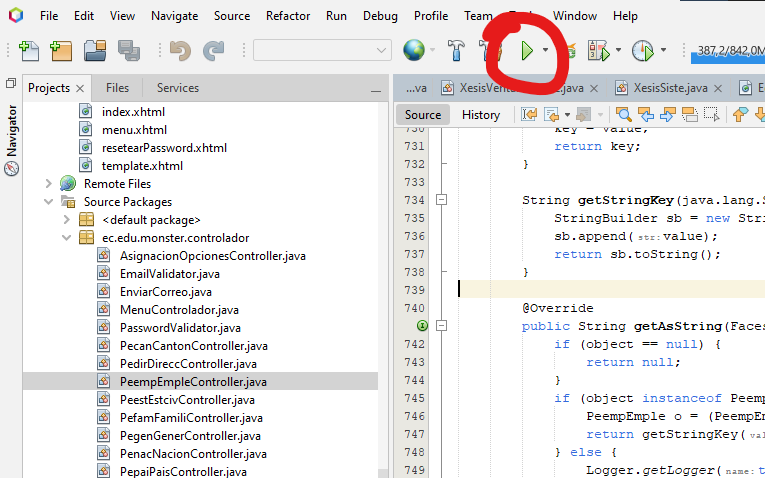


Figura 23. Botón para ejecutar el proyecto.

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

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Figura 24. Ejecución del proyecto.

Al ingresar con las credenciales y al dirigirse al apartado de Procesos > Seguridad > Usuarios por Perfil se visualizará la siguiente pantalla.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 25. Usuarios por Perfil.

Al seleccionar algún rol aparecerán los usuarios pertenecientes a ducho perfil y con los botones del centro se puede cambiar de rol a dicho usuario.

Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 26. Asignación de un usuario a un perfil.

# 6. CONCLUSIONES

* La asignación de permisos de forma dinámica en una aplicación web hecha con Java EE y PrimeFaces ofrece una mayor flexibilidad y seguridad en la gestión de usuarios y roles.
* La implementación de esta funcionalidad puede requerir un esfuerzo adicional en el diseño y desarrollo de la aplicación, pero puede resultar en una mayor facilidad de mantenimiento y escalabilidad.
* Es importante establecer una estrategia clara para la gestión de permisos, incluyendo la definición de roles y la asignación de permisos a nivel de usuario y grupo, para garantizar que la aplicación cumpla con los requisitos de seguridad y privacidad.

# 7. RECOMENDACIONES

* Se recomienda utilizar un enfoque basado en roles para la asignación de permisos, lo que permite una mayor granularidad y flexibilidad en la gestión de permisos.
* Es importante establecer un proceso de revisión y auditoría regular para la gestión de permisos, a fin de garantizar que los usuarios y roles tengan los permisos adecuados y que se cumplan los requisitos de seguridad y privacidad.
* Se recomienda utilizar herramientas y frameworks de seguridad bien establecidos y probados para la implementación de la asignación de permisos dinámica, como Spring Security o Java EE Security, para garantizar la integridad y la confidencialidad de los datos y sistemas.

# 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] | «¿Qué es el perfil del usuario? Y ¿Para qué sirve?,» IBM, [En línea]. Available: https://tinyurl.com/vcznat7d. [Último acceso: 10 03 2023]. |
| [13] | «Roles de base de datos,» IBM, 17 03 2021. [En línea]. Available: https://tinyurl.com/5bky97tb. [Último acceso: 23 01 2023]. |
| [14] | H. Mejía y A. Montero, «Php Progra33,» [En línea]. Available: https://phpprogra33.webcindario.com/Fundamentos%20P.O.O.html. [Último acceso: 8 Febrero 2023]. |
| [15] | «Desarrollo Web,» 8 Abril 2014. [En línea]. Available: https://desarrolloweb.com/articulos/herencia-en-programacion-orientada-objetos.html. [Último acceso: 8 Febrero 2023]. |
| [16] | IBM, «IBM,» 14 Abril 2021. [En línea]. Available: https://www.ibm.com/docs/es/i/7.3?topic=information-smtp. |
| [17] | Felipe, «HostingPlus,» 8 Marzo 2021. [En línea]. Available: https://www.hostingplus.pe/blog/protocolo-pop-que-es-y-como-funciona/. |
| [18] | J. Jimenez, «Redes Zone,» 2022 Diciembre 28. [En línea]. [Último acceso: 2023]. |