MANUAL TÉCNICO.

Nombre: Fredy José Gabriel Herrera Funes. No. Carné: 202130478

-Descripción: aplicación web sobre revistas, teniendo roles de administrador, anunciante, editor, suscriptor, con la posibilidad de crear y publicar revistas y capitulos pdf.

-Herramientas utilizadas:

- Desarrollado en sistema operativo Ubunto linux, versión 24.04.
- IDE utilizado: Apache NetBeans 22.
- Servidor para base de datos: MySQL.
- Servidor Apache tomcat 10.
- Lenguaje de programación utilizado: Java (Programación orientada a objetos), JavaScript, HTML.
- Versión Java: 17.

-Explicación del código:

- **Backend:**

1. La clase AdministradorRevista está diseñada para gestionar la publicación de revistas. Se encarga de recibir datos desde una solicitud HTTP, validar la información y almacenar una nueva revista en la base de datos.

2. Atributos

DBRevistas db: Una instancia de la clase DBRevistas, que probablemente maneja la conexión y las operaciones de la base de datos relacionadas con las revistas.

3. Método publicarRevista

Este método se encarga de procesar una solicitud de publicación de revista. Aquí están los pasos que sigue:

Obtener la sesión:

Utiliza request.getSession(false) para obtener la sesión actual sin crear una nueva. Esto es importante para asegurarse de que se está trabajando con una sesión válida.

Extraer el usuario:

Se obtiene el objeto Usuario almacenado en la sesión, que contiene información sobre el usuario que está realizando la acción. Se extrae el nombre de usuario (userName).

Recoger etiquetas:

Se obtienen las etiquetas seleccionadas desde la solicitud HTTP. Si se seleccionan etiquetas válidas (definidas en el enum Etiquetas), se añaden a una lista; si alguna etiqueta es inválida, se registra un error.
Crear la revista:

Se instancian y configuran los atributos de un objeto Revista utilizando los parámetros de la solicitud, como el nombre, la descripción, la categoría, las etiquetas, la fecha de publicación, y algunos comentarios y "me gusta".
Establecer el autor:

Se asigna el nombre de usuario como autor de la revista.
Verificación de existencia:

Se llama al método verificarRevistaExistente para asegurarse de que no haya una revista con el mismo nombre en la base de datos.
Guardar la revista:

Finalmente, si todo está en orden, se llama al método guardarRevista de DBRevistas para almacenar la nueva revista en la base de datos.
4. Método verificarRevistaExistente
Este método se utiliza para comprobar si ya existe una revista con el nombre proporcionado. Si es así, lanza una excepción UserDataException con un mensaje indicando que el nombre ya está en uso. Esto ayuda a evitar duplicados en la base de datos.

Conclusión
La clase AdministradorRevista actúa como un controlador en el patrón MVC (Modelo-Vista-Controlador) para la gestión de revistas en una aplicación web. Su función principal es asegurar que los datos de las revistas sean válidos y únicos antes de guardarlos en la base de datos. Es una implementación típica en aplicaciones que manejan datos ingresados por el usuario y necesitan validar y almacenar esa información de manera segura.

1. La clase AdministradorPassword está diseñada para gestionar la seguridad de las contraseñas en una aplicación. Vamos a desglosar sus componentes y funcionalidades:

El objetivo principal de esta clase es proporcionar métodos para encriptar contraseñas y verificar si una contraseña ingresada coincide con una contraseña previamente encriptada.

2. Métodos
hashPassword
java
Copiar código
```
public String hashPassword(String password) {
    return BCrypt.hashpw(password, BCrypt.gensalt(12));
}
```
Descripción: Este método recibe una contraseña en texto plano y devuelve su versión encriptada.
Funcionalidad:
Utiliza BCrypt.hashpw para encriptar la contraseña.
BCrypt.gensalt(12) genera una sal con un factor de costo de 12. Este factor determina cuántas veces se aplica el algoritmo de hashing, lo que hace que el proceso sea más lento y, por lo tanto, más seguro contra ataques de fuerza bruta.
checkPassword
java
Copiar código
```
public boolean checkPassword(String password, String hashedPassword) {
    return BCrypt.checkpw(password, hashedPassword);
}
```
Descripción: Este método verifica si una contraseña ingresada coincide con una contraseña encriptada.
Funcionalidad:
Utiliza BCrypt.checkpw para comparar la contraseña en texto plano con la versión encriptada.
Devuelve true si la contraseña coincide y false en caso contrario.

3. Importancia de la clase
Seguridad: Utiliza el algoritmo BCrypt, que es un estándar robusto para el almacenamiento seguro de contraseñas. Este enfoque ayuda a proteger las contraseñas de los usuarios contra ataques como la inyección de datos o el acceso no autorizado.
Facilidad de uso: Proporciona una interfaz sencilla para gestionar la encriptación y la verificación de contraseñas, lo que facilita su implementación en diversas partes de una aplicación.

Conclusión
La clase AdministradorPassword es una implementación esencial para la gestión de contraseñas en una aplicación segura. Su uso de BCrypt garantiza que las contraseñas se manejen de manera segura, ofreciendo protección contra diversas amenazas de seguridad.

1. La clase AdministradorUsuarios está diseñada para gestionar las operaciones relacionadas con los usuarios en una aplicación. A continuación, te explico sus componentes y funcionalidades:

Propósito de la clase
El objetivo de esta clase es permitir la creación, validación y gestión de usuarios en un sistema, interactuando con una base de datos a través de la clase DBRevistas.

2. Atributos
DBRevistas db: Instancia de la clase DBRevistas, que se encarga de las operaciones de acceso a datos relacionadas con los usuarios.
boolean exist: Variable que se utiliza para comprobar si un usuario ya existe (aunque parece no ser necesaria, ya que se puede manejar directamente en el flujo).

3. Métodos
crearUsuario
java
Copiar código
```
public Usuario crearUsuario(HttpServletRequest req) throws UserDataException {
    Usuario newUser = validarDatos(req);

    if (db.usuarioExistente(newUser.getUserName())) {
        exist = true;
        throw new UserDataException("¡El nombre de usuario ya existe!");
    }
    db.guardarUsuario(newUser);
    return newUser;
}
```
Descripción: Este método crea un nuevo usuario.
Funcionalidad:
Llama al método validarDatos para obtener un nuevo objeto Usuario a partir de los datos de la solicitud HTTP.
Verifica si el nombre de usuario ya existe utilizando usuarioExistente.
Si el usuario existe, lanza una excepción UserDataException.
Si el usuario no existe, guarda el nuevo usuario en la base de datos y lo devuelve.
validarLogin
java
Copiar código
```
public boolean validarLogin(String userName, String password) {
    return db.verificarUsuario(userName, password);
```

}
Descripción: Verifica las credenciales de inicio de sesión del usuario.
Funcionalidad:
Llama a verificarUsuario, que se encarga de validar el nombre de usuario y la contraseña en la base de datos.
Devuelve true si las credenciales son correctas, y false en caso contrario.
validarDatos
java
Copiar código

```
private Usuario validarDatos(HttpServletRequest req) throws UserDataException {
    Usuario newUser = new Usuario();
    try {
        newUser.setUserName(req.getParameter("userName"));
        newUser.setPassword(req.getParameter("password"));
        newUser.setRol(Roles.valueOf(req.getParameter("rol")));
        newUser.setCartera(Float.parseFloat(req.getParameter("cartera")));
    } catch (NullPointerException | IllegalArgumentException e) {
        throw new UserDataException("Error en los datos enviados");
    }

    if (newUser.esValido()) {
        return newUser;
    }
    throw new UserDataException("Error en los datos enviados");
}
```

Descripción: Valida y construye un nuevo objeto Usuario a partir de los datos de la solicitud.
Funcionalidad:
Extrae los parámetros necesarios del objeto HttpServletRequest.
Intenta establecer los valores correspondientes en el objeto Usuario.
Maneja excepciones si los datos son nulos o si hay un problema en la conversión.
Verifica si el usuario es válido a través del método esValido(), y si no, lanza una excepción.
obtenerUsuario
java
Copiar código

```
public Usuario obtenerUsuario(String userName) {
    return db.obtenerUsuarioPorNombre(userName);
}
```

Descripción: Obtiene un usuario específico de la base de datos por su nombre de usuario.

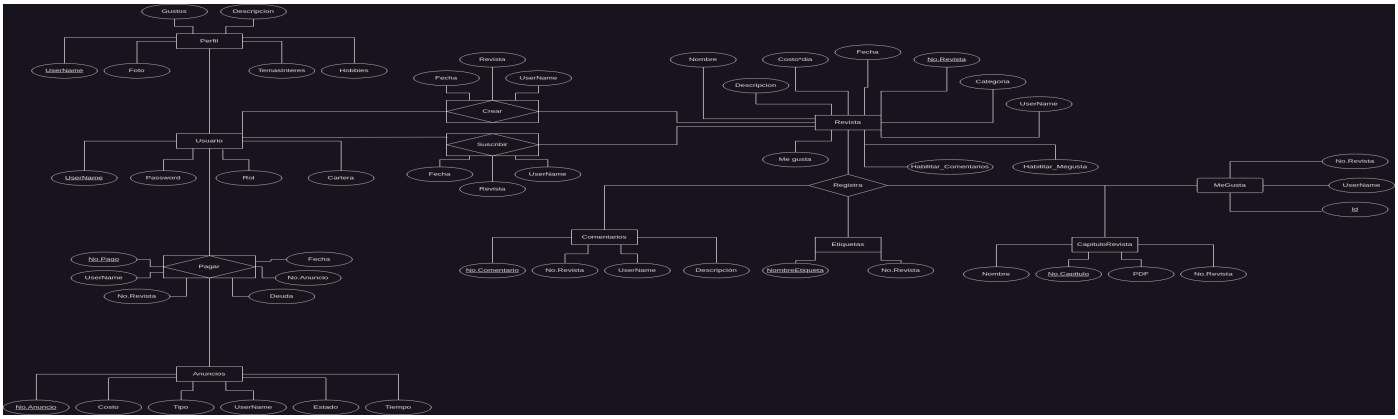
Funcionalidad: Llama al método obtenerUsuarioPorNombre de DBRevistas para recuperar los datos del usuario.

Conclusión
La clase AdministradorUsuarios proporciona una interfaz para la gestión de usuarios en una aplicación, manejando tanto la creación de nuevos usuarios como la validación de credenciales para el inicio de sesión. Utiliza la clase DBRevistas para interactuar con la base de datos, asegurando que las operaciones de usuarios se realicen de manera segura y eficiente. Su implementación de validaciones ayuda a garantizar la integridad de los datos ingresados por los usuarios.

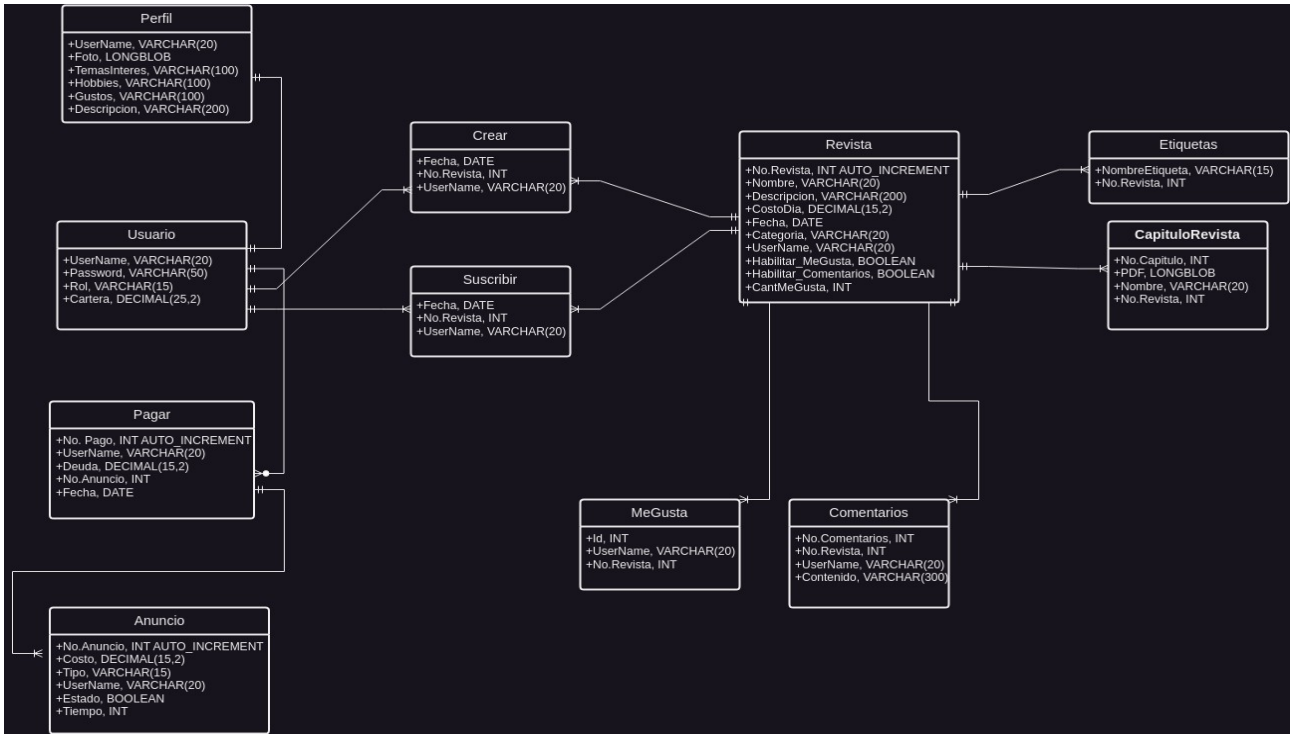
-Adicional: tambien se utilizaron clase para el manejo y conexión con la base de datos, y tambien se manejaron clases tipo servlets para el manejo y conexión del fronted.

-Diagramas:

-Diagrama Entidad/Relación:



-Diagrama de Tablas:



-Mapeo Físico de la Base de Datos:

```
CREATE SCHEMA CONTROL_SISTEMA_REVISTAS;

CREATE TABLE usuario (
    user_name VARCHAR(20) NOT NULL,
    password VARCHAR(70) NOT NULL,
    rol VARCHAR(15) NOT NULL,
    cartera DECIMAL(25,2) NOT NULL,
    CONSTRAINT PK_USUARIO PRIMARY KEY (user_name)
);

CREATE TABLE perfil (
    user_name VARCHAR(20) NOT NULL,
    foto LONGBLOB,
    tema_interes VARCHAR(100) NOT NULL,
    hobbies VARCHAR(100) NOT NULL,
    gustos VARCHAR(100) NOT NULL,
    descripcion VARCHAR(200) NOT NULL,
    CONSTRAINT PK_PERFIL PRIMARY KEY (user_name),
        CONSTRAINT FK_USUARIO_IN_USER_NAMEP FOREIGN KEY
(user_name) REFERENCES usuario(user_name)
);
```

```
CREATE TABLE anuncio (
    no_anuncio INT AUTO_INCREMENT NOT NULL,
    costo DECIMAL(15,2) NOT NULL,
    user_name VARCHAR(20) NOT NULL,
    tipo VARCHAR(20) NOT NULL,
    contenido LONGBLOB NOT NULL,
    estado BOOLEAN NOT NULL,
    tiempo INT NOT NULL,
    CONSTRAINT PK_NO_ANUNCIO PRIMARY KEY (no_anuncio),
            CONSTRAINT   FK_USUARIO_IN_USER_NAME   FOREIGN   KEY
(user_name) REFERENCES usuario(user_name)
);

CREATE TABLE pago (
    no_pago INT AUTO_INCREMENT NOT NULL,
    user_name VARCHAR(20) NOT NULL,
    deuda DECIMAL(15,2) NOT NULL,
    no_anuncio INT NOT NULL,
    fecha DATE NOT NULL,
    CONSTRAINT PK_PAGO PRIMARY KEY (no_pago),
            CONSTRAINT   FK_USUARIO_IN_USER_NAME   FOREIGN   KEY
(user_name) REFERENCES usuario(user_name),
          CONSTRAINT   FK_USUARIO_IN_NO_ANUNCIO   FOREIGN   KEY
(no_anuncio) REFERENCES anuncio(no_anuncio)
);

CREATE TABLE revista (
    nombre VARCHAR(20) NOT NULL,
    descripcion VARCHAR(200) NOT NULL,
    costo_dia DECIMAL(15,2),
    fecha DATE NOT NULL,
    categoria VARCHAR(20) NOT NULL,
    etiquetas VARCHAR(255),
    autor VARCHAR(20) NOT NULL,
    habilitar_megusta BOOLEAN NOT NULL,
    habilitar_comentarios BOOLEAN NOT NULL,
    habilitar_suscripciones BOOLEAN NOT NULL,
    cantidad_megusta INT ,
    CONSTRAINT PK_REVISTA PRIMARY KEY (nombre),
      CONSTRAINT  FK_USUARIO_IN_USER_NAME  FOREIGN  KEY  (autor)
REFERENCES usuario(user_name)
);
```

```
CREATE TABLE suscribir(
   Fecha DATE NOT NULL,
   nombre_revista VARCHAR(20) NOT NULL,
   user_name VARCHAR(20) NOT NULL,
        CONSTRAINT PK_SUSCRIBIR PRIMARY KEY (nombre_revista, user_name),
             CONSTRAINT FK_REVISTA_IN_NOMBRE FOREIGN KEY (nombre_revista) REFERENCES revista(nombre),
          CONSTRAINT FK_USUARIO_IN_USER_NAME FOREIGN KEY (user_name) REFERENCES usuario(user_name)
);

CREATE TABLE me_gusta(
   id INT AUTO_INCREMENT NOT NULL,
   user_name VARCHAR(20) NOT NULL,
   nombre_revista VARCHAR(20) NOT NULL,
   CONSTRAINT PK_ME_GUSTA PRIMARY KEY (id),
            CONSTRAINT FK_REVISTA_IN_NOMBRER FOREIGN KEY (nombre_revista) REFERENCES revista(nombre),
         CONSTRAINT FK_USUARIO_IN_USER_NAMEU FOREIGN KEY (user_name) REFERENCES usuario(user_name)
);

CREATE TABLE comentario(
   no_comentario INT AUTO_INCREMENT NOT NULL,
   user_name VARCHAR(20) NOT NULL,
   nombre_revista VARCHAR(20) NOT NULL,
   contenido VARCHAR(300) NOT NULL,
   CONSTRAINT PK_COMENTARIO PRIMARY KEY (no_comentario),
            CONSTRAINT FK_REVISTA_IN_NOMBREE FOREIGN KEY (nombre_revista) REFERENCES revista(nombre),
            CONSTRAINT FK_USUARIO_INUSERNAME FOREIGN KEY (user_name) REFERENCES usuario(user_name)
);

CREATE TABLE capitulo_revista(
   no_capitulo INT AUTO_INCREMENT NOT NULL,
   nombre_revista VARCHAR(20) NOT NULL,
   pdf LONGBLOB NOT NULL,
   CONSTRAINT PK_CAPITULO_REVISTA PRIMARY KEY (no_capitulo),
           CONSTRAINT FK_REVISTA_IN_NO_REVISTA FOREIGN KEY (nombre_revista) REFERENCES revista(nombre),
);
```