



## **INSTITUTO TECNOLÓGICO DE CIUDAD MADERO**

**Departamento de Sistemas y Computación**

**Ingeniería en Sistemas Computacionales**

### **Manual Técnico:**

## **PROTOTIPO DE UN SISTEMA DE BASE DE DATOS COMO REPOSITORIO PARA HERRAMIENTAS TIC's PARA EL SECTOR MINERO**

### **Alumno:**

González Guerrero Carlos Alberto

18070070

[L18070070@cdmadero.tecnm.mx](mailto:L18070070@cdmadero.tecnm.mx)

Tel: (833) 442 1606

### **Asesor Externo:**

M.C. Martha Laura Chuey Rubio

**Depto. Sistemas y Computación**

[martha.cr@cdmadero.tecnm.mx](mailto:martha.cr@cdmadero.tecnm.mx)

Tel: (833) 357 4820 ext: 3021

### **Empresa:**

Instituto Tecnológico de Ciudad Madero

Avenida Primero de Mayo 1610.

Los Mangos, CP. 89460.

Ciudad Madero, Tamaulipas.

Tel: (833) 357 4820

### **Asesor Interno:**

M.C. Rosa Delia Rétiz Rivera

**Depto. Sistemas y Computación**

[rosa.rr@cdmadero.tecnm.mx](mailto:rosa.rr@cdmadero.tecnm.mx)

Tel: (833) 357 4820

**Fecha de inicio:** 2 de Febrero del 2023  
**Fecha de terminación:** 2 de Junio del 2023  
**Horario:** 10:00 a.m – 16:00 p.m  
**Total de horas:** 500

## ÍNDICE

<b>1. INTRODUCCIÓN .....</b>	<b>1</b>
<b>2. OBJETIVO .....</b>	<b>1</b>
<b>3. REQUERIMIENTOS MÍNIMOS .....</b>	<b>1</b>
3.1. XAMPP .....	1
3.2. Repositorio de Minería .....	1
3.3. JDK 16.01 .....	1
<b>4. INSTALACIÓN DE SOFTWARE .....</b>	<b>3</b>
4.1 Procedimiento de instalación del repositorio .....	3
4.2 Procedimiento de instalación de XAMPP .....	3
<b>5. ¿CÓMO ABRIR EL SOFTWARE? .....</b>	<b>12</b>
5.1 XAMPP y phpMyAdmin .....	12
5.1.1. Solución a error de conexión con el servicio de MySQL .....	13
5.1.2. Permitir el guardar imágenes mayores a 1 MB .....	19
5.1.3. Importar bases de datos mayores a 20 MB .....	21
5.1.4. Creación de usuario (OBLIGATORIO) .....	25
5.2 Procedimiento de instalación de JDK 16.01 .....	27
<b>6. LIBRERÍAS USADAS .....</b>	<b>30</b>
6.1. mysql-connector-java-8.0.25 .....	30
6.2 JCalendar 1.4 .....	30
<b>7. ESTRUCTURA DE LA BASE DE DATOS .....</b>	<b>31</b>
<b>8. CLASES .....</b>	<b>32</b>
8.1 Clase Main .....	32
8.2 Clases de Apoyo .....	33
8.2.1 Conexion .....	33
8.2.2. Areas .....	33
8.2.3 Consultas .....	34
8.2.4 Simuladores .....	36
8.2.5 TablaImagen .....	37
8.2.6 Validaciones .....	37
8.3 Clases Gráficas .....	39



8.3.1. InicioSesion .....	40
8.3.2. AgregarUsuarios .....	43
8.3.3. MainScreenFrame .....	47
8.3.4 DatabaseMenuPanel .....	52
8.3.5. AgregarSimuladorPanel .....	58
8.3.6. VerSimuladorFrame .....	64
8.3.7. AgregarArea .....	76
8.3.8. DatosPersonales .....	84
8.3.9. VerUsuariosPanel .....	96

## 1. INTRODUCCIÓN

Este proyecto se ha realizado con el fin de centralizar la información referente a distintos simuladores computacionales de minería, para tenerlos como un punto de referencia al momento de que la empresa cliente quiera desarrollar su propio simulador en el futuro próximo.

Este sistema se programó en Java usando NetBeans, y MySQL Workbench en conjunto con phpMyAdmin para elaborar la base de datos.

## 2. OBJETIVO

El objetivo de este manual es mostrar los datos técnicos en cuanto al sistema desarrollado, en sí para facilitar la modificación o actualizaciones del mismo en caso de que así sea necesario, o bien para el mantenimiento posterior del mismo.

## 3. REQUERIMIENTOS MÍNIMOS

### 3.1. XAMPP

**Procesador:** Intel Celeron (32 o 64 bits)

**Memoria RAM:** 256 MB

**Espacio en disco:** 85 MB de espacio libre en el disco

### 3.2. Repositorio de Minería

**Procesador:** Celeron (32 o 64 bits)

**Memoria RAM:** 2 GB de RAM

**Espacio en disco:** 10 MB

**Gráficos:** Intel HD Graphics

### 3.3. JDK 16.01

**Procesador:** Celeron (32 o 64 bits)

**Memoria RAM:** 2 GB de RAM

**Espacio en disco:** 300 MB

**Gráficos:** Intel HD Graphics

## 4. INSTALACIÓN DE SOFTWARE

### 4.1 Procedimiento de instalación del repositorio

El programa y todo lo necesario para su ejecución se encuentra dentro de la carpeta llamada “**Repositorio**”. El ejecutable es portable, por lo que no es necesario instalar el y puede ejecutarse en casi cualquier computadora con Windows 7 en adelante.

Nombre	Fecha de modificación	Tipo	Tamaño
imagenes	30/12/2021 08:15 a. m.	Carpeta de archivos	
JRE	30/12/2021 08:15 a. m.	Carpeta de archivos	
lib	30/12/2021 08:15 a. m.	Carpeta de archivos	
Repositorio	27/12/2021 03:15 a. m.	Aplicación	5,646 KB
repositoriomineria	30/12/2021 09:54 a. m.	SQL Text File	12,113 KB
xampp-windows-x64-8.0.7-0-VS16-instal...	25/06/2021 06:43 p. m.	Aplicación	161,589 KB

### 4.2 Procedimiento de instalación de XAMPP

#### Paso 1 (OPCIONAL): Descarga

**Nota:** No es necesario realizar la instalación de XAMPP si usted sólo planea conectarse a través de una computadora dentro de la misma red, pero, si usted quiere usar la base de datos de forma por su cuenta, sin necesidad de conectarse a Internet y a otra computadora, será necesario que siga estos pasos.

#### Paso 1 (OPCIONAL): Descarga

El instalador de XAMPP ya se encuentra en la carpeta “Repositorio”, pero si desea descargarlo

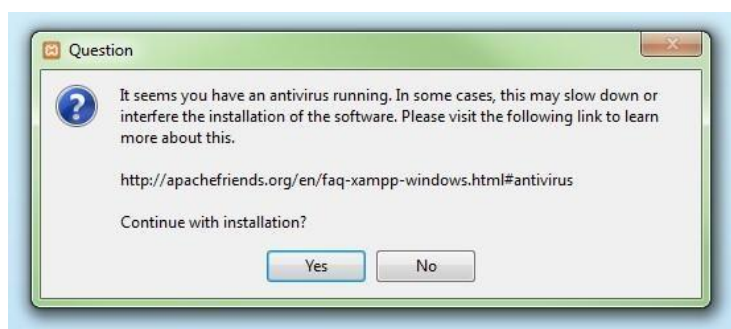
por su cuenta, las versiones con PHP 5.5, 5.6 o 7 se pueden descargar gratuitamente desde la página del proyecto [Apache Friends](#).

## **Paso 2: Ejecutar el archivo .exe**

Una vez descargado el paquete, puedes ejecutar el archivo [.exe](#) haciendo doble clic en él.

## **Paso 3 (OPCIONAL): Desactivar el programa antivirus**

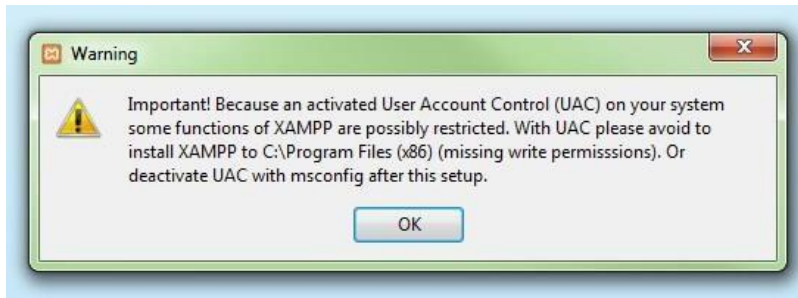
Se recomienda desactivar el programa antivirus hasta que todos los componentes estén instalados, ya que puede obstaculizar el proceso de instalación.



Antes de iniciar la instalación de XAMPP es recomendable desactivar temporalmente el antivirus

## **Paso 4 (OPCIONAL): Desactivar el UAC**

También el control de cuentas de usuario (User Account Control, UAC) puede interferir en la instalación, ya que limita los derechos de escritura en la unidad de disco C:\. Para saber cómo desactivar temporalmente el UCA puedes dirigirte a las páginas de [soporte de Microsoft](#).



También el Control de cuentas de usuarios (UAC) puede impedir la instalación de XAMPP.



## Paso 5: Iniciar el asistente de instalación

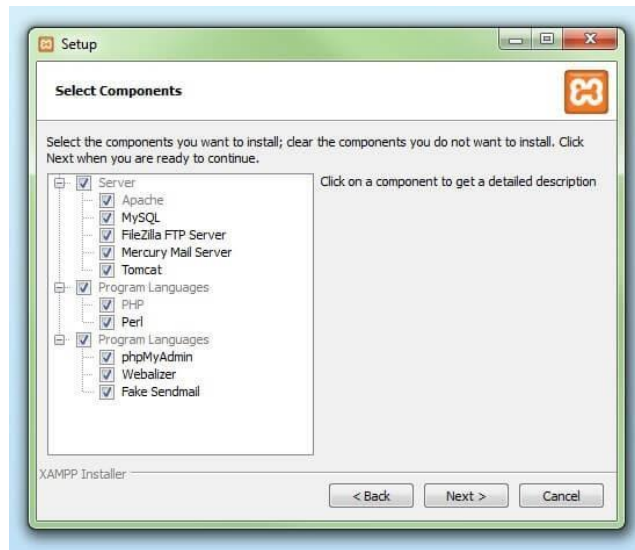
Una vez superados estos pasos, aparece la pantalla de inicio del asistente para instalar XAMPP. Para ajustar las configuraciones de la instalación se hace clic en “Next”.



Con la aparición de la pantalla de inicio del asistente da comienzo la instalación de XAMPP

## Paso 6: Selección de los componentes del software

En la rúbrica “Select components” se pueden excluir de la instalación componentes aislados del paquete de software de XAMPP. Se recomienda la configuración estándar para un servidor de prueba local, con la cual se instalan todos los componentes disponibles. Confirma la selección haciendo clic en “Next”.

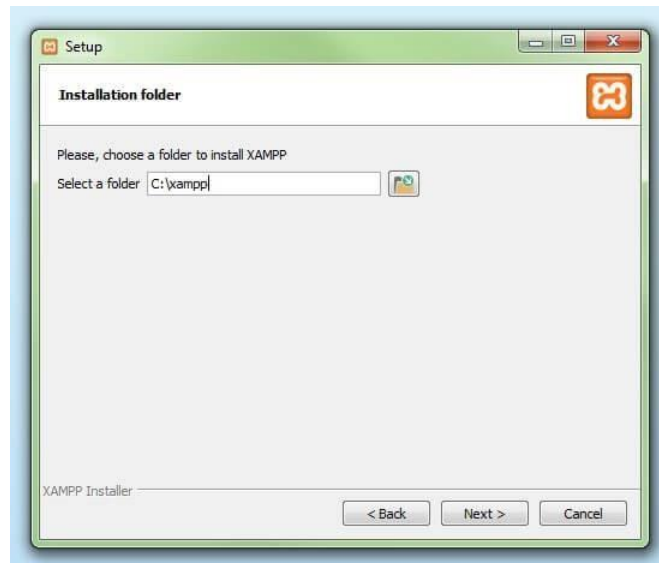


En el cuadro de diálogo "Select Components" se pueden seleccionar o deseleccionar los componentes que se instalarán.

## Paso 7: Selección del directorio para la instalación

En este paso se escoge el directorio donde se instalará el paquete. Si se ha escogido la configuración estándar se creará una carpeta con el nombre XAMPP en C:\.

En un siguiente paso, se selecciona el directorio donde se instalarán los archivos.



## Paso 8: Iniciar el proceso de instalación

El asistente extrae los componentes seleccionados y los guarda en el directorio escogido en un proceso que puede durar algunos minutos. El avance de la instalación se muestra como una barra de carga de color verde.



A continuación, da comienzo el proceso de instalación en el cual se descomprimen los elementos de software seleccionados y se instalan en el directorio que se ha definido en los preajustes.

## Paso 9: Configurar Firewall

Durante el proceso de instalación es frecuente que el asistente avise del bloqueo de Firewall. En la ventana de diálogo puedes marcar las casillas correspondientes para permitir la comunicación del servidor Apache en una red privada o en una red de trabajo. Recuerda que no se recomienda usarlo en una red pública.

Durante la instalación será necesario reconfigurar el cortafuegos para que no bloquee componentes del servidor Apache.



## Paso 10: Cerrar la instalación

Una vez extraídos e instalados todos los componentes puedes cerrar el asistente con la tecla “Finish”. Para acceder inmediatamente al panel de control solo es necesario marcar la casilla que pregunta si deseamos hacerlo.



Haciendo clic en “Finish”, se cierra el asistente de instalación de XAMPP.

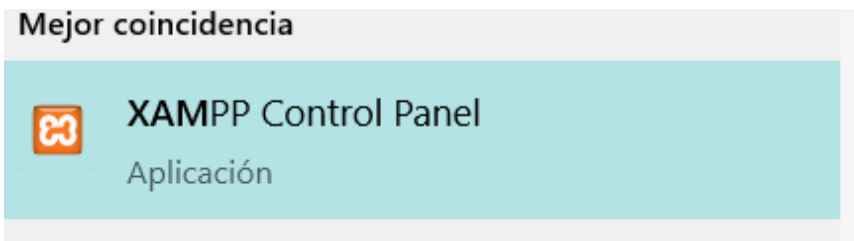
Una vez completados estos pasos, su computadora podrá funcionar para conectarse de forma local a la base de datos.

## 5. ¿CÓMO ABRIR EL SOFTWARE?

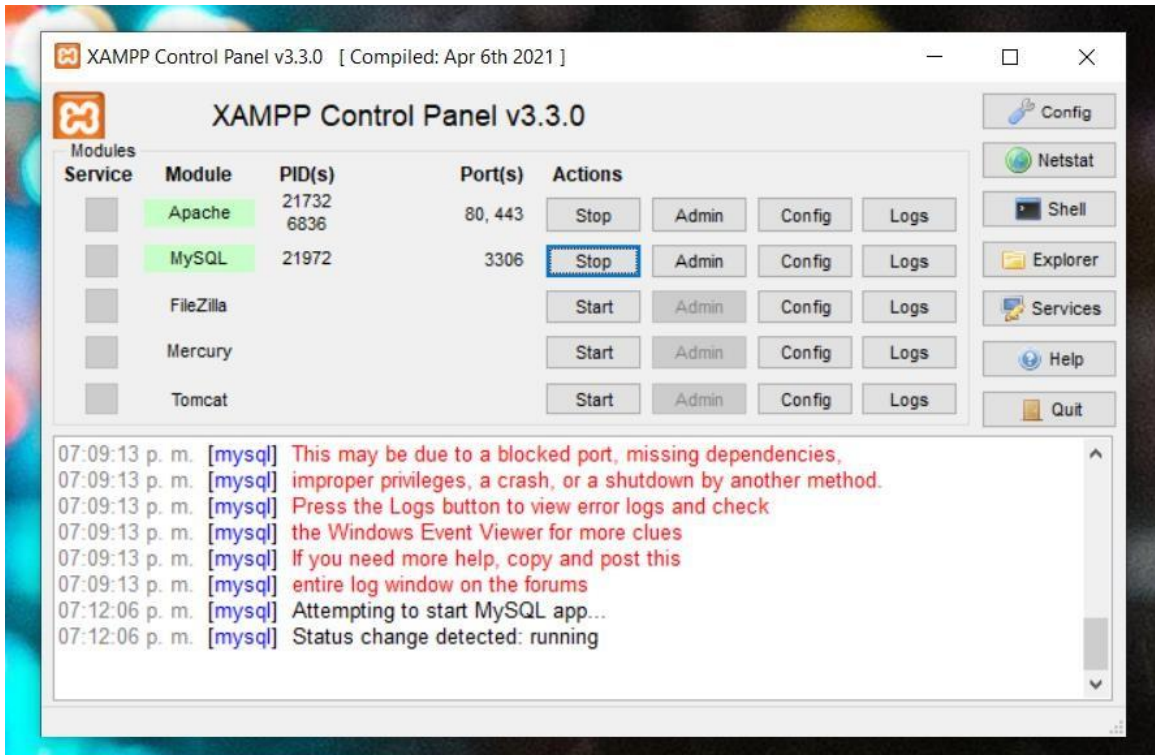
### 5.1 XAMPP y phpMyAdmin

Explicación detallada del proceso para abrir el software y hacer la conexión con la base de datos mediante XAMPP.

1. Antes de iniciar a ejecutar el programa haremos la conexión con XAMPP de Apache y MySQL, para esto abrimos el programa.



2. Una vez dentro daremos clic en Start en Apache y MySQL hasta que estos se iluminen de verde, se establezca una conexión correcta



### 5.1.1. Solución a error de conexión con el servicio de MySQL

Probablemente pudiéramos tener problemas al momento de iniciar los servicios, en específico el servicio de MySQL que sería lo siguiente:



XAMPP Control Panel v3.2.2 [ Compiled: Nov 12th 2015 ]

### XAMPP Control Panel v3.2.2

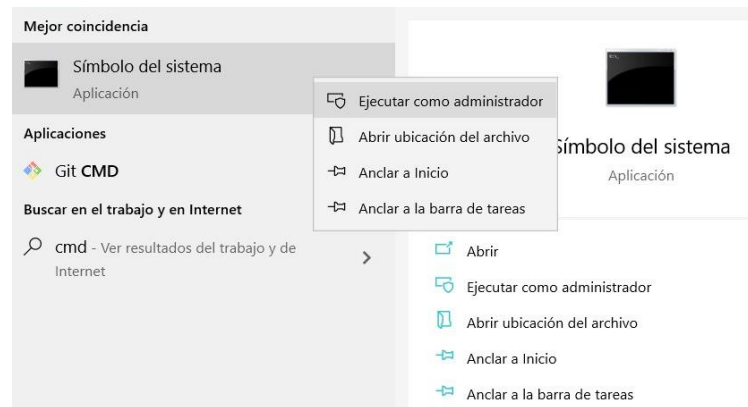
Service	Module	PID(s)	Port(s)	Actions
<input type="checkbox"/>	Apache	4504 7400	80, 443	<input type="button" value="Stop"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	MySQL			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	FileZilla			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	Mercury			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>
<input type="checkbox"/>	Tomcat			<input type="button" value="Start"/> <input type="button" value="Admin"/> <input type="button" value="Config"/> <input type="button" value="Logs"/>

```

20:27:04 [mysql] Attempting to start MySQL app...
20:27:05 [Apache] Status change detected: running
20:27:05 [mysql] Status change detected: running
20:27:08 [mysql] Status change detected: stopped
20:27:08 [mysql] Error: MySQL shutdown unexpectedly.
                This may be due to a blocked port, missing dependencies,
                improper privileges, a crash, or a shutdown by another method.
                Press the Logs button to view error logs and check
                the Windows Event Viewer for more clues
                If you need more help, copy and post this
                entire log window on the forums

```

Como se puede observar no aparece en verde como Apache, esto se debe comúnmente debido ya se había instalado MySQL en esa computadora previamente. Para solucionar esto nos iremos al buscador de Windows y escribimos “cmd”, y aquí en símbolo de sistema daremos clic derecho y clic en “Ejecutar como administrador”.



Nos abrirá la siguiente ventana de comando en la cual escribiremos lo siguiente “net stop MySQL” y presionamos el botón Enter.

```

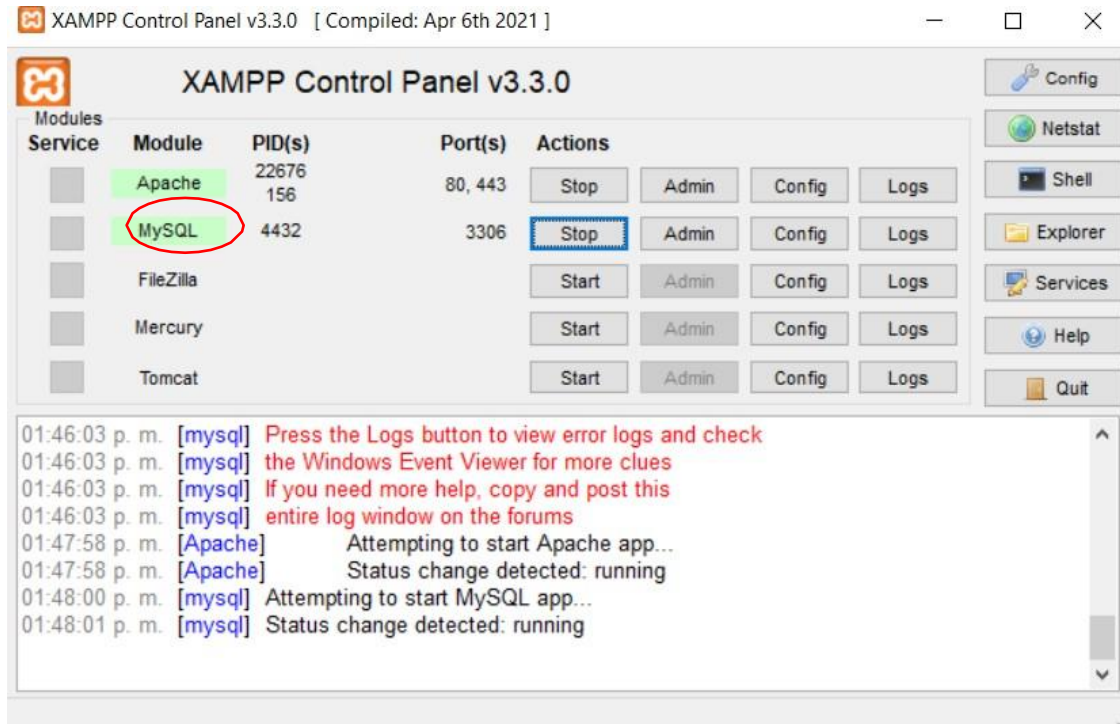
Administrador: Símbolo del sistema

Microsoft Windows [Versión 10.0.19042.1052]
(c) Microsoft Corporation. Todos los derechos reservados.

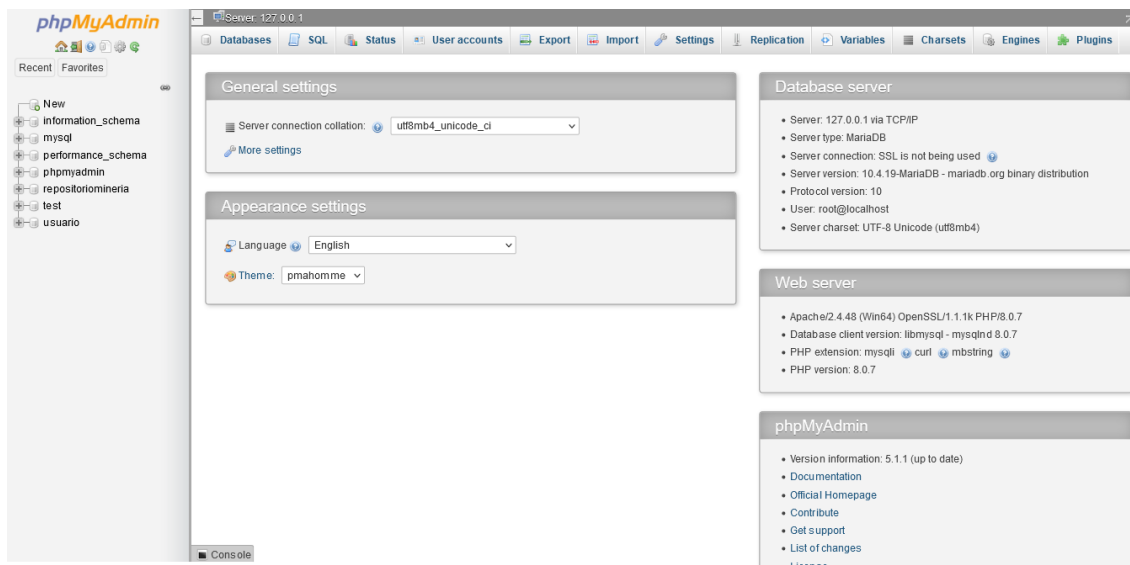
C:\WINDOWS\system32>net stop MySQL80
El servicio de MySQL80 está deteniéndose.
El servicio de MySQL80 se detuvo correctamente.

C:\WINDOWS\system32>
  
```

Y ahora regresamos a XAMPP y volvemos a intentar iniciar el servicio de MySQL, y como se observa ya nos deja iniciar dicho servicio.



- Daremos clic en el botón Admin de MySQL, el cual nos redirige a phpMyAdmin en nuestro navegador principal.



- Para importar la base de datos debemos crear un schema con el mismo nombre del archivo. SQL e importarlo. Esto se logra presionando “New” en la barra izquierda y asignándole el mismo nombre del archivo, el cual es “repositoriomineria” con el formato “utf8mb4\_general\_ci”.

## Databases

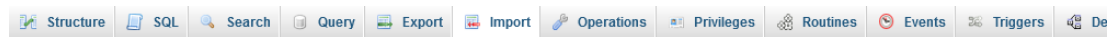
Create database

repositoriomineria

utf8mb4\_general\_ci

Create

Una vez creado, debes seleccionar “**Import**” en la barra superior de phpMyAdmin, presionar el botón “**Browse**” y seleccionar el archivo .sql que se encuentra en la carpeta del programa. Una vez seleccionado el archivo, se presiona el botón “**Go**” en la parte inferior **sin modificar ningún otro valor en esta pantalla.**



### Importing into the database "repositoriomineria"

#### File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.  
A compressed file's name must end in .[format].[compression].

Browse your computer: [Browse...](#) No file selected.

You may also drag and drop a file on any page.

Character set of the file: utf-8

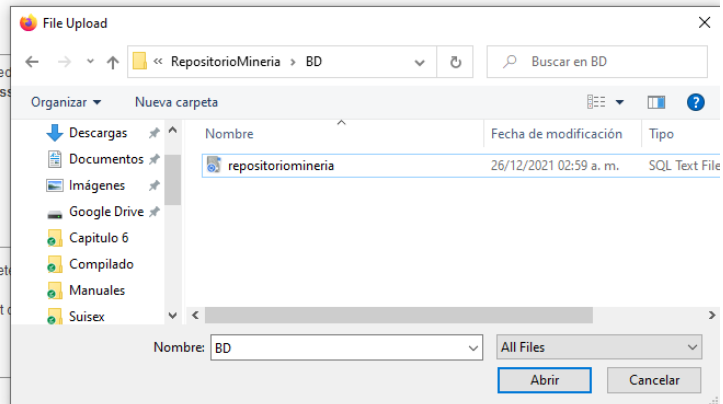
#### Partial import:

☒ Allow the interruption of an import in case the script detects an error.

Skip this number of queries (for SQL) starting from the first one.

#### Other options:

☒ Enable foreign key checks



- Una ventana similar a la siguiente nos indicará que la importación fue la correcta.

Server: 127.0.0.1 > Database: repositoriomineria

Structure SQL Search Query Export Import Operations Privileges Routines Events Triggers Designer

✔ Import has been successfully finished, 36 queries executed. (repositoriomineria.sql)

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0032 seconds.)

```
-- phpMyAdmin SQL Dump -- version 5.1.1 -- https://www.phpmyadmin.net/ -- Host: 127.0.0.1 -- Generation Time: Dec 30, 2021 at 04:54 PM -- Server version: 10.4.19-MariaDB -- PHP Version: 8.0.7
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
START TRANSACTION;
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
SET time_zone = "+00:00";
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

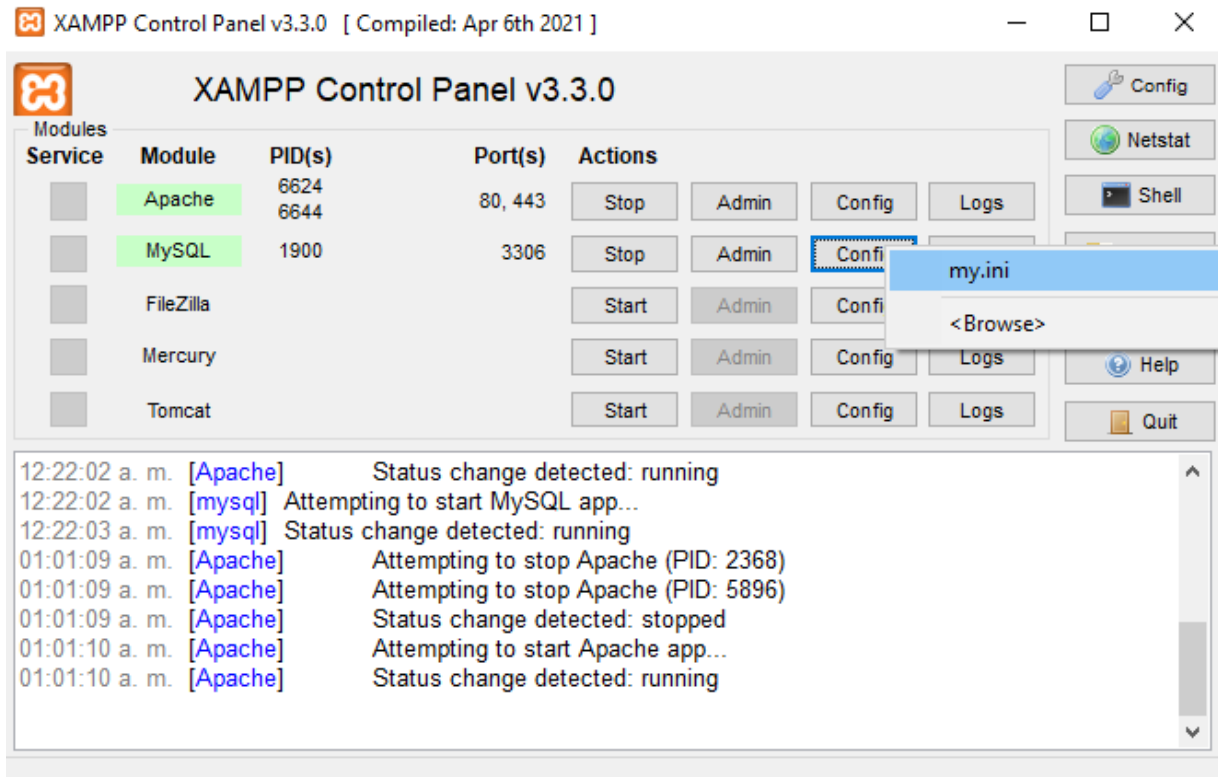
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
```

[ Edit inline ] [ Edit ] [ Create PHP code ]

### 5.1.2. Permitir el guardar imágenes mayores a 1 MB

Por defecto, el sistema sólo permitirá guardar imágenes 1 MB o menos. Para cambiar esto, primero presione el botón “Config” de MySQL. Saldrá una lista de opciones. Presione “my.ini” y abrirá un documento de texto.



Cerca del inicio del archivo, en la sección **[mysqld]**, línea 37, cambie el valor de la variable “max\_allowed\_packet” a 512 MB o cuantos MBs quiera. **Consejo: para buscar más fácilmente en el Bloc de Notas, presione Control + B y seleccione la casilla “Ajuste automático”, de esta forma podrá buscar las variables más fácilmente.**

my: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
# Here follows entries for some specific programs

# The MySQL server
default-character-set=utf8mb4
[mysqld]
port=3306
socket="D:/Programas/xampp/mysql/mysql.sock"
basedir="D:/Programas/xampp/mysql"
tmpdir="D:/Programas/xampp/tmp"
datadir="D:/Programas/xampp/mysql/data"
pid_file="mysql.pid"
# enable-named-pipe
key_buffer=16M
max_allowed_packet=512M
sort_buffer_size=512K
net_buffer_length=8K
read_buffer_size=256K
read_rnd_buffer_size=512K
myisam_sort_buffer_size=8M
log_error="mysql_error.log"

# Change here for bind listening
# bind-address="127.0.0.1"
```

Línea 37, columna 1 100% Windows (CRLF) UTF-8

También, en la sección **[mysqldump]**, línea 163, cambie el valor de la variable “max\_allowed\_packet” por un valor de su preferencia.

```
my: Bloc de notas
Archivo Edición Formato Ver Ayuda
innodb_lock_wait_timeout=50

## UTF 8 Settings
#init-connect='SET NAMES utf8\'
#collation_server=utf8_unicode_ci
#character_set_server=utf8
#skip-character-set-client-handshake
#character_sets_dir="D:/Programas/xampp/mysql/share/charsets"
sql_mode=NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGINE_SUBSTITUTION
log_bin_trust_function_creators=1

character-set-server=utf8mb4
collation-server=utf8mb4_general_ci
[mysqldump]
max_allowed_packet=512M

[mysql]
# Remove the next comment character if you are not familiar with SQL
#safe-updates

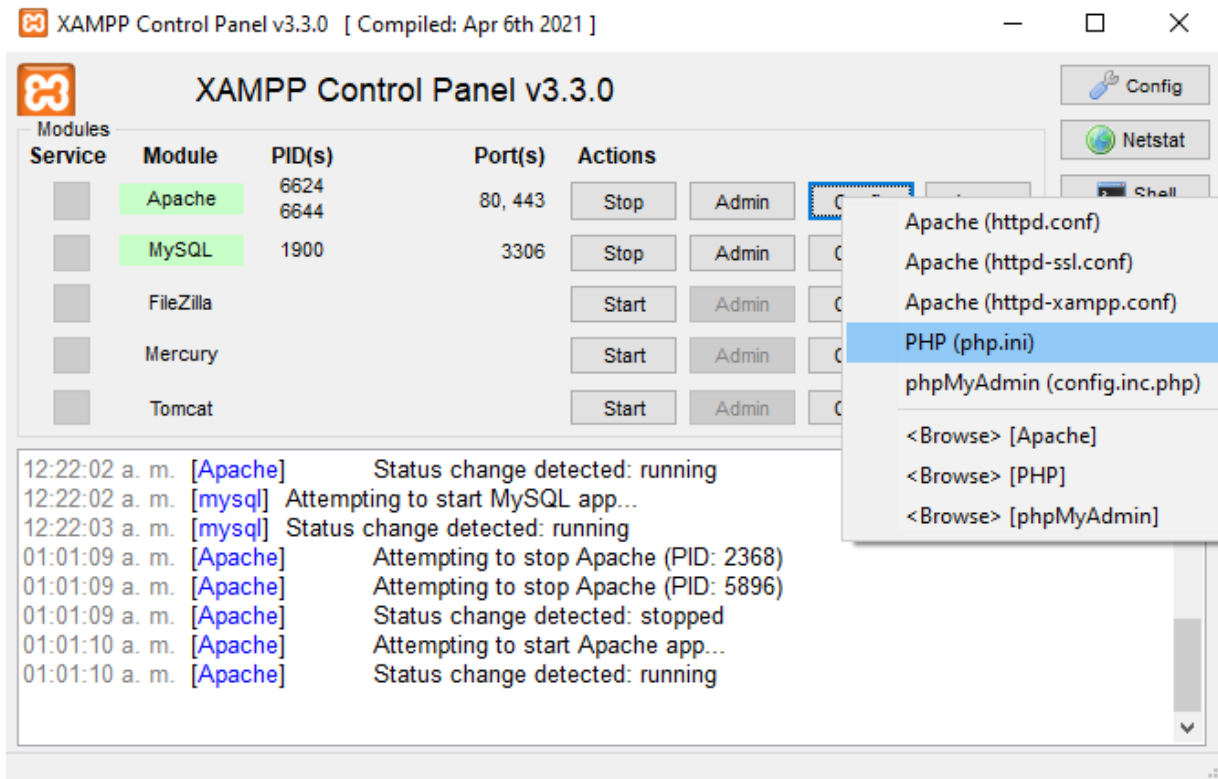
[isamchk]
key_buffer=20M
sort_buffer_size=20M
read_buffer=2M
write_buffer=2M

Línea 163, columna 19 100% Windows (CRLF) UTF-8
```

### 5.1.3. Importar bases de datos mayores a 20 MB.

Por defecto, phpMyAdmin sólo permite importar archivos .sql de 20 MB (40 MB en algunas versiones). Para cambiar esto, se deberán cambiar algunos valores en el archivo php.ini. Para hacer esto, primero presiona el botón “Config” de Apache en XAMPP. Esto abrirá una lista de opciones; selecciona “PHP (php.ini)”. Esto abrirá un documento de texto.





Primero, cambia el valor de la variable “max\_execution\_time” en la línea 407 a 300. **Consejo:** para buscar más fácilmente en el Bloc de Notas, presione Control + B y seleccione la casilla “Ajuste automático”, de esta forma podrá buscar las variables más fácilmente.

```
php: Bloc de notas
Archivo Edición Formato Ver Ayuda
; Default Value: 15
; Development Value: 15
; Production Value: 0
zend.exception_string_param_max_len=15

;;;;;;;;;;;;;;;;
; Miscellaneous ;
;;;;;;;;;;;;;;;;

; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header). It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; http://php.net/expose-php
expose_php=On

;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;

; Maximum execution time of each script, in seconds
; http://php.net/max-execution-time
; Note: This directive is hardcoded to 0 for the CLI SAPI
max_execution_time=300

<
Siguiendo encontrado desde la parte inferior Línea 407, columna 19 100% Windows (CRLF) UTF-8
```

Después, busque la variable “upload\_max\_filesize” en la línea 863 a 128MB (o cualquier otro valor de MBs su preferencia).

```
php: Bloc de notas
Archivo Edición Formato Ver Ayuda

; cgi.check_shebang_line controls whether CGI PHP checks for line starting with #!
; (shebang) at the top of the running script. This line might be needed if the
; script support running both as stand-alone script and via PHP CGI<. PHP in CGI
; mode skips this line and ignores its content if this directive is turned on.
; http://php.net/cgi.check-shebang-line
cgi.check_shebang_line=1

;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

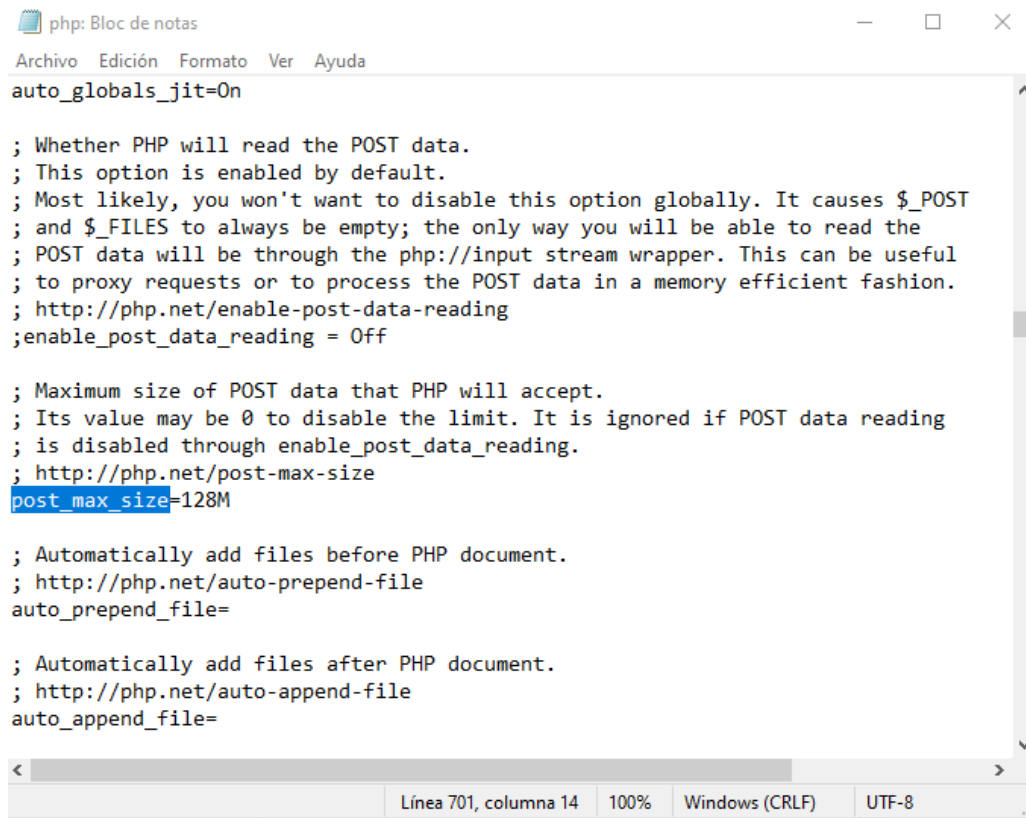
; Whether to allow HTTP file uploads.
; http://php.net/file-uploads
file_uploads=On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
; http://php.net/upload-tmp-dir
upload_tmp_dir="D:\Programas\xampp\tmp"

; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize=128M

<
Siguiendo encontrado desde la parte inferior Línea 853, columna 20 100% Windows (CRLF) UTF-8
```

Seguido de esto, busque la variable “post\_max\_size” en la línea 701 e ingrese **el mismo valor que introdujo en “upload\_max\_size”**.



```
php: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
auto_globals_jit=On

; Whether PHP will read the POST data.
; This option is enabled by default.
; Most likely, you won't want to disable this option globally. It causes $_POST
; and $_FILES to always be empty; the only way you will be able to read the
; POST data will be through the php://input stream wrapper. This can be useful
; to proxy requests or to process the POST data in a memory efficient fashion.
; http://php.net/enable-post-data-reading
;enable_post_data_reading = Off

; Maximum size of POST data that PHP will accept.
; Its value may be 0 to disable the limit. It is ignored if POST data reading
; is disabled through enable_post_data_reading.
; http://php.net/post-max-size
post_max_size=128M

; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file=

; Automatically add files after PHP document.
; http://php.net/auto-append-file
auto_append_file=

Línea 701, columna 14  100%  Windows (CRLF)  UTF-8
```

Por último, cambie el valor de la variable “memory\_limit” en la línea 428 a 512MB.

```
php: Bloc de notas
Archivo Edición Formato Ver Ayuda
max_input_time=60

; Maximum input variable nesting level
; http://php.net/max-input-nesting-level
;max_input_nesting_level = 64

; How many GET/POST/COOKIE input variables may be accepted
;max_input_vars = 1000

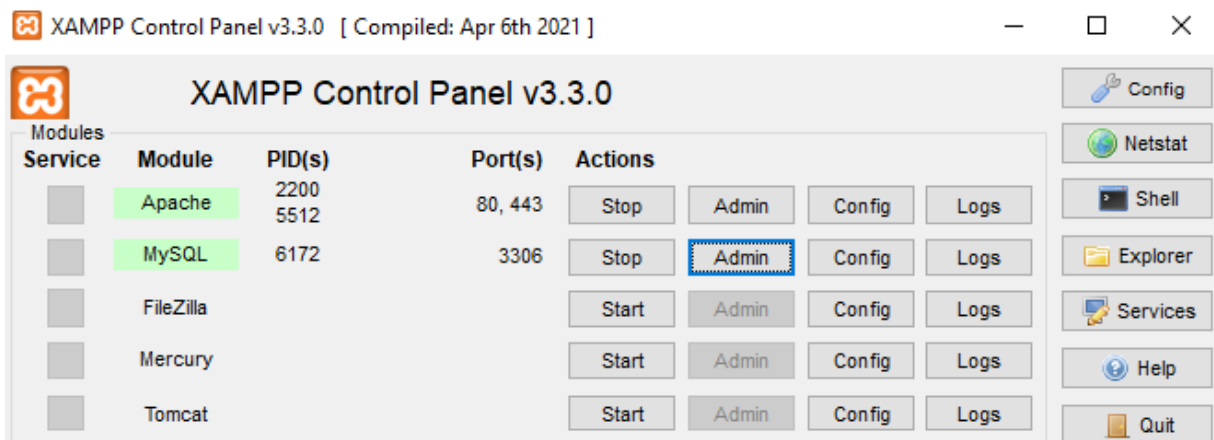
; Maximum amount of memory a script may consume
; http://php.net/memory-limit
memory_limit=512M

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

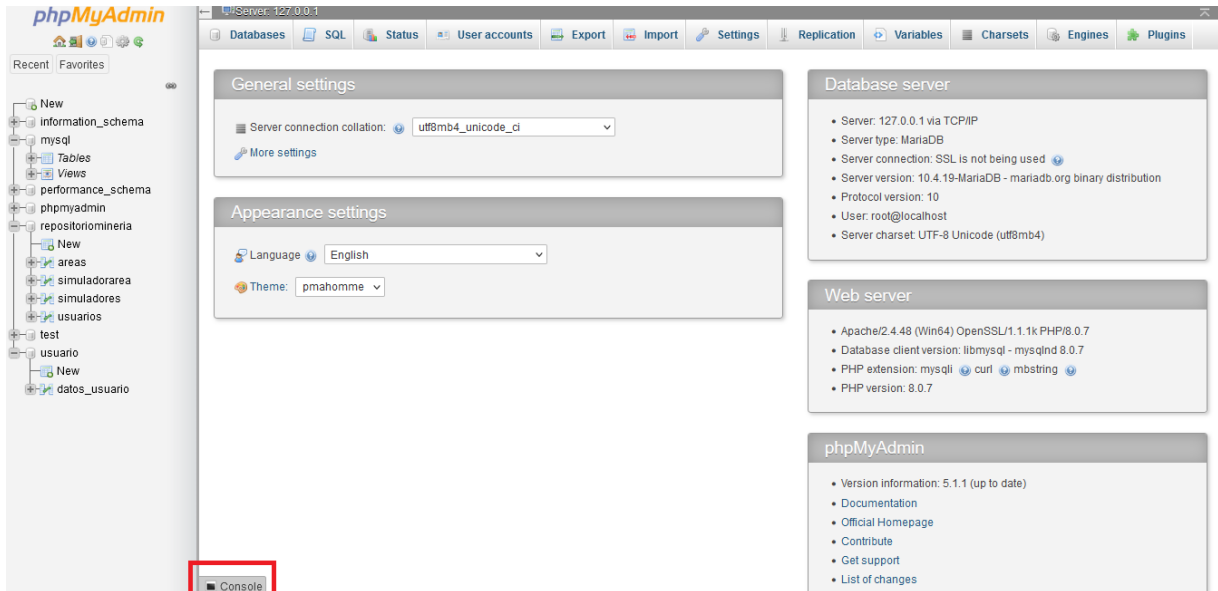
; This directive informs PHP of which errors, warnings and notices you would like
; it to take action for. The recommended way of setting values for this
; directive is through the use of the error level constants and bitwise
; operators. The error level constants are below here for convenience as well as
; some common settings and their meanings.
; By default, PHP is set to take action on all errors, notices and warnings EXCEPT
; those related to E_NOTICE and E_STRICT, which together cover best practices and
; recommended coding standards in PHP. For performance reasons, this is the
<
Siguiente encontrado desde la parte inferior  Línea 428, columna 13  100%  Windows (CRLF)  UTF-8
```

#### 5.1.4. Creación de usuario (OBLIGATORIO)

Antes de poder usar el sistema, deberá crear un nuevo usuario en phpMyAdmin con todos los permisos para poder conectarse a la base de datos. Primero, debe abrir phpMyAdmin a través de XAMPP presionando el botón “Admin” de MySQL.



Una vez dentro de phpMyAdmin, deberá abrir la consola. Esta se encuentra en la esquina inferior izquierda. Deberá presionar en el botón “Console” que se ve en la siguiente imagen:



Dentro de ella, deberá escribir exactamente los siguientes comandos como se muestran en la imagen:

```

Console

Press Ctrl+Enter to execute query

>CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
>GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' WITH GRANT OPTION;
>CREATE USER 'username'@'%' IDENTIFIED BY 'password';
>GRANT ALL PRIVILEGES ON *.* TO 'username'@'%' WITH GRANT OPTION;
>FLUSH PRIVILEGES;
    
```

Deberá escribirlos línea por línea, presionando Ctrl + Enter para ejecutar cada línea. Aquí está la lista de comandos que debe copiar y pegar:

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'username'@'localhost' WITH GRANT OPTION;
```








```
CREATE USER 'username'@'%' IDENTIFIED BY 'password';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'username'@'%' WITH GRANT OPTION;
```

```
FLUSH PRIVILEGES;
```

## 5.2 Procedimiento de instalación de JDK 16.01

El JDK (Java Development Kit 16.01) se puede encontrar en la carpeta “**Repositorio**”.

	imagenes	30/12/2021 08:15 a. m.	Carpeta de archivos	
	JRE	30/12/2021 08:15 a. m.	Carpeta de archivos	
	lib	30/12/2021 08:15 a. m.	Carpeta de archivos	
	jdk-16.0.1_windows-x64_bin	25/06/2021 04:59 p. m.	Aplicación	154,174 KB
	Repositorio	27/12/2021 03:15 a. m.	Aplicación	5,646 KB
	repositoriomineria	30/12/2021 09:54 a. m.	SQL Text File	12,113 KB
	xampp-windows-x64-8.0.7-0-VS16-instal...	25/06/2021 06:43 p. m.	Aplicación	161,589 KB

Sin embargo, si desea buscarlo por su propia cuenta o posee un sistema operativo diferente, puede descargarlo directamente desde [la página oficial de Oracle](#).

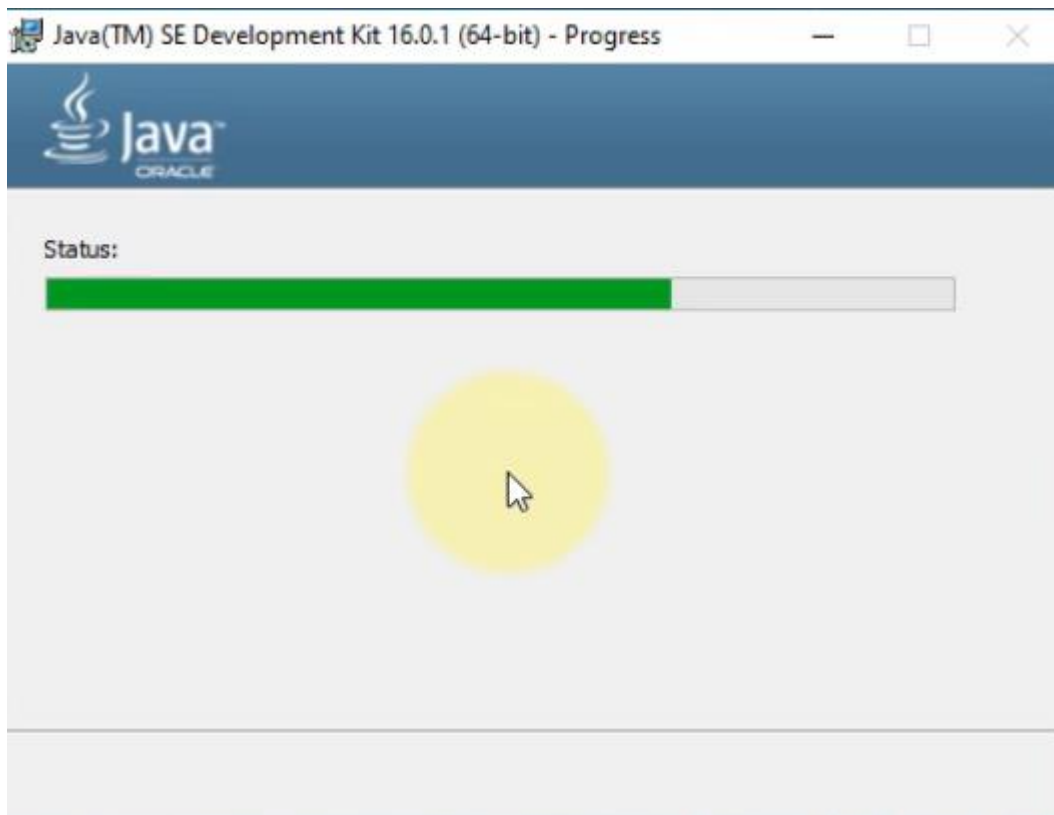
Debe darle doble click al archivo “**jdk-16.01\_windows-x64.bin**”, lo cual hará que le aparezca la siguiente pantalla:



Presiona “Next” y saldrá la siguiente pantalla:



Aquí usted puede seleccionar la ruta de instalación presionando el botón “Change”. Una vez esté conforme con su elección, presione “Next”.



Comenzará la instalación. Esto puede tomar unos cuantos minutos en completar.





Una vez pasados unos minutos, el proceso de instalación habrá sido completado y podrá hacer cambios al proyecto usando alguna IDE como NetBeans (recomendado), Eclipse u otros.

## 6. LIBRERÍAS USADAS

### 6.1. mysql-connector-java-8.0.25

MySQL proporciona conectividad para aplicaciones cliente Java con MySQL Connector/J, un controlador que implementa la API de Java Database Connectivity (JDBC). La API es el estándar de la industria para la conectividad independiente de la base de datos entre el lenguaje de programación Java y una amplia gama de bases de datos SQL, hojas de cálculo, etc.

La API de JDBC puede hacer lo siguiente:

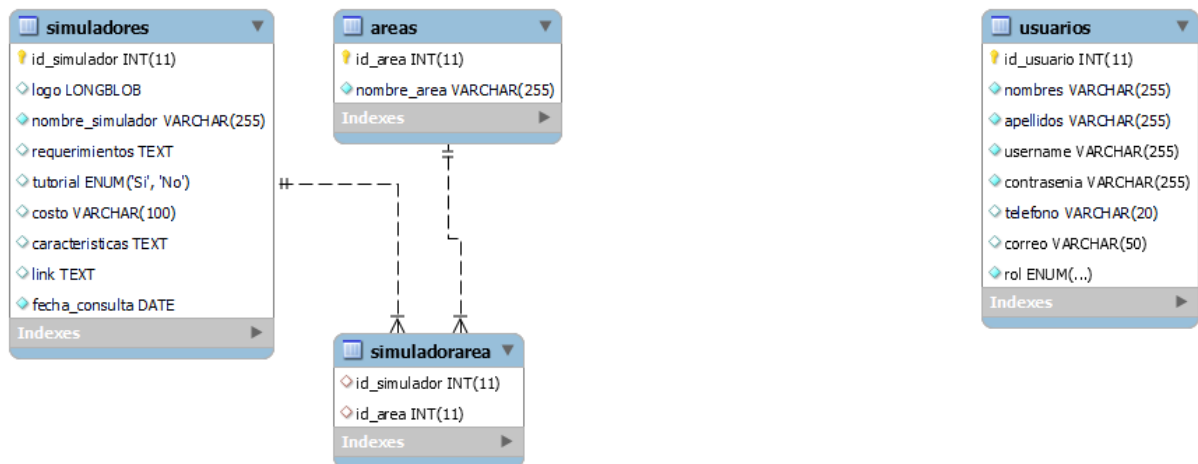
- Establecer una conexión con una base de datos u obtener acceso a cualquier origen de datos tabular.
- Enviar instrucciones SQL.
- Recuperar y procesar los resultados recibidos de la base de datos.

### 6.2 JCalendar 1.4

JCalendar es un selector de fecha en Java para seleccionar gráficamente una fecha. JCalendar está compuesto de varios otros Java beans, un JDayChooser, un JMonthChooser y un JYearChooser. Todos estos beans tienen una propiedad locale, proporcionan varios iconos (Color  $16 \times 16$ , Color  $32 \times 32$ , Mono  $16 \times 16$  y Mono  $32 \times 32$ ) y su propio editor de propiedades locales. Así que pueden ser fácilmente utilizados en constructores de GUI. También parte del paquete es un JDateChooser, un bean compuesto por un JDateEditor (para la edición directa de fecha) y un botón para abrir un JCalendar para seleccionar la fecha.

May						2006	
	Sun	Mon	Tue	Wed	Thu	Fri	Sat
18		1	2	3	4	5	6
19	7	8	9	10	11	12	13
20	14	15	16	17	18	19	20
21	21	22	23	24	25	26	27
22	28	29	30	31			

## 7. ESTRUCTURA DE LA BASE DE DATOS



Como se puede ver en la imagen, la base de datos está conformada por 4 tablas:

- Simuladores: Tabla en donde se guardan los datos de los simuladores.

- Áreas: Tabla en donde se guardan los nombres de las áreas.
- Simuladorarea: Tabla que une a Simuladores y Áreas en una relación M:N.
- Usuarios: Tabla donde se guardan los datos de los usuarios.

## 8. CLASES

En este apartado se explicarán las clases que conforman el sistema y lo que hacen. En total hay 15 clases, 6 de las cuales se clasifican como de “**Apoyo**”, 8 son “**Gráficas**”, y una es la clase Main.

### 8.1 Clase Main

La clase Main lo único que hace es crear una instancia de la clase gráfica “**InicioSesión**”.

```
1  package repositoriomineria;
2
3  import gui.InicioSesionFrame;
4
5  /**
6   *
7   * @author Carlos Alberto Gonzalez Guerrero
8   */
9  public class RepositorioMineria {
10
11     /**
12      * @param args the command line arguments
13      */
14     public static void main(String[] args) {
15         new InicioSesionFrame().setVisible(true);
16     }
17
18 }
```

## 8.2 Clases de Apoyo

Las clases de “apoyo” son aquellas clases que se usan con el fin de apoyar a las clases gráficas, ya sea proporcionando herramientas para conectarse a la base de datos, hacer consultas o modificar las propiedades de algunos elementos gráficos. Son un total de 6: **Areas**, **Conexion**, **Consultas**, **Simuladores**, **TablaImagen** y **Validaciones**.

### 8.2.1 Conexion

Esta es la clase que permite la conexión con la base de datos.

El único método en esta clase es **conectar**, el cual permite la conexión a la base de datos usando como argumentos el nombre del host, de la base de datos, el usuario y la contraseña con el método *getConnection*, el cual es llamado desde la clase *DriveManager*, que es proporcionada por la librería **mysql-connector-java-8.0.27**.

```
public static Connection conectar() {  
    try {  
        Connection cn = DriverManager.getConnection("jdbc:mysql://localhost:3306/repositoriomineria", "root", "");  
        return cn;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return (null);  
}
```

### 8.2.2. Areas

Esta clase está creada con el propósito de obtener el ID de algún área en específico.

Su único método es **getIDArea**, el cual recibe como argumento el nombre de un área y devuelve el ID del área haciendo una consulta a la base de datos.

```
public String getIDArea(String nombre) {  
    String consultaID = "SELECT id_area FROM areas WHERE nombre_area = '" + nombre + "'";  
    try {  
        Connection cn = new Conexion().conectar();  
        PreparedStatement obtenerIDArea = cn.prepareStatement(consultaID);  
        ResultSet rs = obtenerIDArea.executeQuery();  
        while (rs.next()) {  
            id_area = rs.getString("id_area");  
        }  
  
        cn.close();  
        obtenerIDArea.close();  
        rs.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return id_area;  
}
```

### 8.2.3 Consultas

Esta es una clase en donde se obtienen datos de la base de datos misceláneos.

El primer es **getAreas**, que es del tipo *JComboBox*, y se utiliza en algunas clases gráficas como **AgregarSimulador** o **PantallaPrincipal** para mostrar un ComboBox de las áreas en la base de datos.

```
public JComboBox getAreas() {

    JComboBox comboAreas = new JComboBox();
    comboAreas.removeAllItems();

    try {
        Connection cn = new Conexion().conectar();
        PreparedStatement pstAreas = cn.prepareStatement("SELECT nombre_area FROM areas");
        ResultSet rs = pstAreas.executeQuery();

        while (rs.next()) {
            comboAreas.addItem(rs.getString("nombre_area"));
        }

        cn.close();
        pstAreas.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return comboAreas;
}
```

El segundo método es **getListAreas**, el cual, como su nombre indica, devuelve una lista de las áreas en la base de datos.

```
public List getListAreas() {
    List areas = new List();

    try {
        Connection cn = new Conexion().conectar();
        PreparedStatement pstAreas = cn.prepareStatement("SELECT nombre_area FROM areas");
        ResultSet rs = pstAreas.executeQuery();

        while (rs.next()) {
            areas.add(rs.getString("nombre_area"));
        }

        cn.close();
        pstAreas.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return areas;
}
```

El último método es **getLogo**, el cual recibe el id de un simulador y devuelve el logo del

simulador correspondiente.

```
public InputStream getLogo(String id_simulador) {
    InputStream is = null;

    String consulta = "SELECT simuladores.logo "
        + "FROM simuladores "
        + "WHERE simuladores.id_simulador = '" + id_simulador + "' ";

    try {
        Connection cn = new Conexion().conectar();
        PreparedStatement pstConsulta = cn.prepareStatement(consulta);
        ResultSet rs = pstConsulta.executeQuery();

        while (rs.next()) {
            is = rs.getBinaryStream("logo");
        }

        cn.close();
        pstConsulta.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return is;
}
```

#### 8.2.4 Simuladores

Esta es una clase similar a **Areas**, pues su función es casi idéntica, nada más que enfocada a la tabla Simuladores.

Su único método, **getIDSimulador**, recibe el nombre de un simulador y devuelve la ID de este:

```
public String getIDSimulador(String nombre) {
    String consultaID = "SELECT id_simulador FROM simuladores WHERE nombre_simulador = '" + nombre + "'";
    try {
        Connection cn = new Conexion().conectar();
        PreparedStatement obtenerIDSimulador = cn.prepareStatement(consultaID);
        ResultSet rs = obtenerIDSimulador.executeQuery();
        while (rs.next()) {
            id_simulador = rs.getString("id_simulador");
        }

        cn.close();
        obtenerIDSimulador.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return id_simulador;
}
```

### 8.2.5 TablaImagen

Esta es una clase que permite el poder mostrar imágenes en la tabla que se muestra en **PantallaPrincipal**, y de esta forma poder mostrar los logos de cada simulador. Es una clase hija de *DefaultTableCellRenderer*, y hace override al método **getTableCellRendererComponent**. En este método revisa si el argumento “value” de tipo *Object* que recibe es una instancia de *JLabel*, y si lo es, hace un casting al objeto “value”, y de esta forma poder agregar *JLabels* a la tabla, lo que permitirá el poder mostrar imágenes en la tabla.

```
public class TablaImagen extends DefaultTableCellRenderer{

    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
        boolean hasFocus, int row, int column) {

        if(value instanceof JLabel){
            JLabel lbl = (JLabel) value;
            return lbl;
        }

        return super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
    }
}
```

### 8.2.6 Validaciones

Esta es una clase llena de Regexes, los cuales permiten validar que los correos, contraseñas y



números telefónicos sean válidos de acuerdo a las reglas establecidas.

El primer regex, “VALID\_EMAIL\_ADDRESS\_REGEX”, busca por correos del tipo [x@x.x](mailto:x@x.x), por lo que, por ejemplo, un correo como: [pruebadecorreo@live.com](mailto:pruebadecorreo@live.com).

```
public static final Pattern VALID_EMAIL_ADDRESS_REGEX =
    Pattern.compile("^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}$", Pattern.CASE_INSENSITIVE);
```

El segundo regex, “VALID\_PASSWORD\_REGEX”, verifica que la contraseña ingresada cumpla con las siguientes reglas:

1. Debe haber mínimo un dígito.
2. Debe haber mínimo una letra minúscula.
3. Debe haber mínimo una letra mayúscula.
4. No debe haber espacios en blanco.
5. Debe tener mínimo 8 caracteres.

```
public static final Pattern VALID_PASSWORD_REGEX =
    Pattern.compile("(?=.*[0-9]) (?=.*[a-z]) (?=.*[A-Z]) (?!\\S+$) .{8,}$");
```

El tercer regex, “VALID\_PHONE\_REGEX”, permite números telefónicos que contengan ladas internacionales o ladas nacionales, aceptando teléfonos como:

1. (52) 833 111 2345
2. (833) 111 2345
3. 833 111 2345
4. 8331112345
5. 833-111-2345
6. Etc.

Esta clase posee 3 métodos, **validateEmail**, **validatePassword** y **validatePhone**, los cuales hacen, en esencia, lo mismo: reciben una cadena, llaman al método *matcher* de la clase *Matcher* usando el regex correspondiente y regresan un valor booleano que indica si la cadena enviada es válida o no.

```
public static boolean validateEmail(String emailStr) {  
    Matcher matcher = VALID_EMAIL_ADDRESS_REGEX.matcher(emailStr);  
    return matcher.find();  
}  
  
public static boolean validatePassword(String passwordStr) {  
    Matcher matcher = VALID_PASSWORD_REGEX.matcher(passwordStr);  
    return matcher.find();  
}  
  
public static boolean validatePhone(String phoneStr) {  
    Matcher matcher = VALID_PHONE_REGEX.matcher(phoneStr);  
    return matcher.find();  
}
```

### 8.3 Clases Gráficas

Estas son las clases que manejan las interfaces del sistema. Las clases que conforman esta categoría son: **AgregarAreasPanel**, **AgregarSimuladorPanel**, **AgregarUsuarios**, **DatosPersonalesPanel**, **InicioSesionFrame**, **MainScreenFrame**, **VerSimuladorFrame** y **VerUsuariosPanel**.

### 8.3.1. InicioSesion



En esta clase se maneja la pantalla de inicio de sesión.

El primer método de la clase es el constructor:

```
public InicioSesionFrame() {
    initComponents();
    this.setLocationRelativeTo(null);
    this.setResizable(false);
    this.setIconImage(getIconImage());
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

Aquí se inician los componentes gráficos de la pantalla.

El segundo método en esta clase es **getIconImage**, cuya función es obtener la imagen que servirá como ícono.

```
@Override
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit().getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El tercer método es  **jButtonSessionActionPerformed**, el cual es llamado cuando se presiona el botón “Iniciar Sesión”:

```
String user = jTextFieldUser.getText().trim();
String password = jPassword.getText().trim();
```

Inicia obteniendo el texto en los dos campos de texto y guardándolos en las variables “user” y “password”.

```
if(jTextFieldUser.getText().length() == 0 || jPassword.getPassword().length == 0){
    JOptionPane.showMessageDialog(this, "Usuario o contraseña no ingresado");
    jTextFieldUser.setText("");
    jPassword.setText("");
}
```

Se verifica que los campos de nombre de usuario y contraseña no estén vacíos. Si lo están, se manda un mensaje de error. De lo contrario, el método continúa.

```

else{
    try{
        Connection cn = new Conexion(ipHost).conectar();
        PreparedStatement pst = cn.prepareStatement(
            "select id_usuario, username, contrasenia, rol from usuarios where username = '
            + user + "' and contrasenia = '" + password + "'";
        ResultSet rs = pst.executeQuery();

        if(rs.next()){
            String id_usuario = rs.getString("id_usuario");
            String username = rs.getString("username");
            //String contrasenia = rs.getString("contrasenia");
            String rol = rs.getString("rol");
            dispose();
            new PantallaPrincipal(id_usuario, username, rol, ipHost).setVisible(true);
        }
        else{
            jPassword.setText("");
            jTextFieldUser.setText("");
            JOptionPane.showMessageDialog(this, "Usuario o contraseña incorrectos");
        }
        cn.close();
        pst.close();
    }
    catch(Exception e){
        JOptionPane.showMessageDialog(this, "No se puede entablar conexión con el servidor.");
        System.out.println(e);
    }
}

```

Aquí se crea una conexión a la base de datos. Se recolectan los datos del usuario que corresponda a ese nombre de usuario y contraseña. De existir un usuario con esos datos, se inicia sesión, cerrando **InicioSesion** y creando una nueva instancia de **PantallaPrincipal**, mandando el id del usuario, nombre de usuario y rol como argumentos. De lo contrario, se muestra un mensaje diciendo que el usuario o contraseña son incorrectos.

Todo lo anterior se realiza dentro de un try-catch, el cual arrojará un mensaje en caso de que haya un error al intentar entablar conexión con el host, ya sea porque el nombre es incorrecto o porque el host no tiene activado los servicios de Apache y MySQL de XAMPP o porque ambas computadoras no están conectadas a Internet en caso de querer hacer una conexión con dos computadoras.

El último método en la clase es **botonCrearCuentaActionPerformed**, el cual es llamado cuando se presiona el botón “Crear Cuenta”.

```
private void botonCrearCuentaActionPerformed(java.awt.event.ActionEvent evt) {  
    new AgregarUsuariosPanel().setVisible(true);  
}
```

Crea una nueva instancia de **AgregarUsuarios**.

### 8.3.2. AgregarUsuarios

Esta clase maneja la pantalla para registrar un usuario. Tiene un método constructor el cual sólo inicializa la interfaz.

Además, comparte el método **getIconImage**, que sirve para escoger la imagen que servirá como ícono de la pantalla:

El último método, **botonRegistrarseActionPerformed**, se llama cuando se presiona el botón “Registrarse”. Lo primero que hace es recuperar todos los valores ingresados en los campos de texto.

```
try{
    String nombres = textNombres.getText().trim();
    String apellidos = textApellidos.getText().trim();
    String username = textUsername.getText().trim();
    String contrasena = textContrasena.getText().trim();
    String telefono = textTelefono.getText().trim();
    String correo = textCorreo.getText().trim();
```

Después verifica que los campos no estén vacíos. Si no lo están, el método continúa creando varias consultas de apoyo para verificar que varios de los datos ingresados no existan ya en la base de datos:

```
if(nombres.length() != 0 && apellidos.length() != 0 && username.length() != 0 && contrasena.length() != 0
&& telefono.length() != 0 && correo.length() != 0){
    String checkUsername = "SELECT * FROM usuarios WHERE username = '" + username + "'";
    String checkTelefono = "SELECT * FROM usuarios WHERE telefono = '" + telefono + "'";
    String checkCorreo = "SELECT * FROM usuarios WHERE correo = '" + correo + "'";

    Connection cn = new Conexion(ipAddress).conectar();

    PreparedStatement pstCheckUsername = cn.prepareStatement(checkUsername);
    PreparedStatement pstCheckTelefono = cn.prepareStatement(checkTelefono);
    PreparedStatement pstCheckCorreo = cn.prepareStatement(checkCorreo);

    ResultSet rsCheckUsername = pstCheckUsername.executeQuery();
    ResultSet rsCheckTelefono = pstCheckTelefono.executeQuery();
    ResultSet rsCheckCorreo = pstCheckCorreo.executeQuery();
```

Después ejecuta las consultas y hace las siguientes verificaciones en este orden:

1. Que el nombre de usuario no exista en la base de datos.
2. Que el teléfono no exista en la base de datos.
3. Que el teléfono sea válido de acuerdo al regex en la clase **Validaciones**.
4. Que el correo no exista en la base de datos.
5. Que el correo sea válido de acuerdo al regex en la clase **Validaciones**.
6. Que la contraseña sea válida de acuerdo al regex en la clase **Validaciones**.

```
if(!rsCheckUsername.next()){
    if(!rsCheckTelefono.next()){
        if(Validaciones.validatePhone(telefono)){
            if(!rsCheckCorreo.next()){
                if(Validaciones.validateEmail(correo)){
                    if(Validaciones.validatePassword(contrasena)){
```

Si se cumplen con todas esas condiciones, el método continúa y se registra el usuario en la base de datos.

```
String insertUser = "INSERT INTO usuarios values(?,?,?,?,?,?,?,?)";

PreparedStatement pstInsertUser = cn.prepareStatement(insertUser);

pstInsertUser.setString(1, "0");
pstInsertUser.setString(2, nombres);
pstInsertUser.setString(3, apellidos);
pstInsertUser.setString(4, username);
pstInsertUser.setString(5, contrasena);
pstInsertUser.setString(6, telefono);
pstInsertUser.setString(7, correo);
pstInsertUser.setString(8, "Usuario");

pstInsertUser.executeUpdate();

JOptionPane.showMessageDialog(this, "Usuario creado.");

textNombres.setText("");
textApellidos.setText("");
textUsername.setText("");
textContrasena.setText("");
textTelefono.setText("");
textCorreo.setText("");
textCorreo.setText("");

cn.close();
pstInsertUser.close();
```

Caso contrario, se arroja un mensaje dependiendo de qué valor es incorrecto o duplicado:









```
        else{
            JOptionPane.showMessageDialog(this, "Contraseña no válida. Debe haber mínimo 8 "
                + "caracteres, una o más mayúsculas y minúsculas, y uno o más números");
            textContrasena.setText("");
        }
        else{
            JOptionPane.showMessageDialog(this, "Correo no válido.");
            textCorreo.setText("");
        }
    }
    else{
        JOptionPane.showMessageDialog(this, "Ya existe un usuario con este correo.");
        textCorreo.setText("");
    }
}
else{
    JOptionPane.showMessageDialog(this, "Teléfono no válido");
    textTelefono.setText("");
}
}
else{
    JOptionPane.showMessageDialog(this, "Ya existe un usuario con este teléfono.");
    textTelefono.setText("");
}
}
else{
    JOptionPane.showMessageDialog(this, "Ya existe este nombre de usuario.");
    textUsername.setText("");
}
}
```

### 8.3.3. MainScreenFrame

Repositorio de Minería

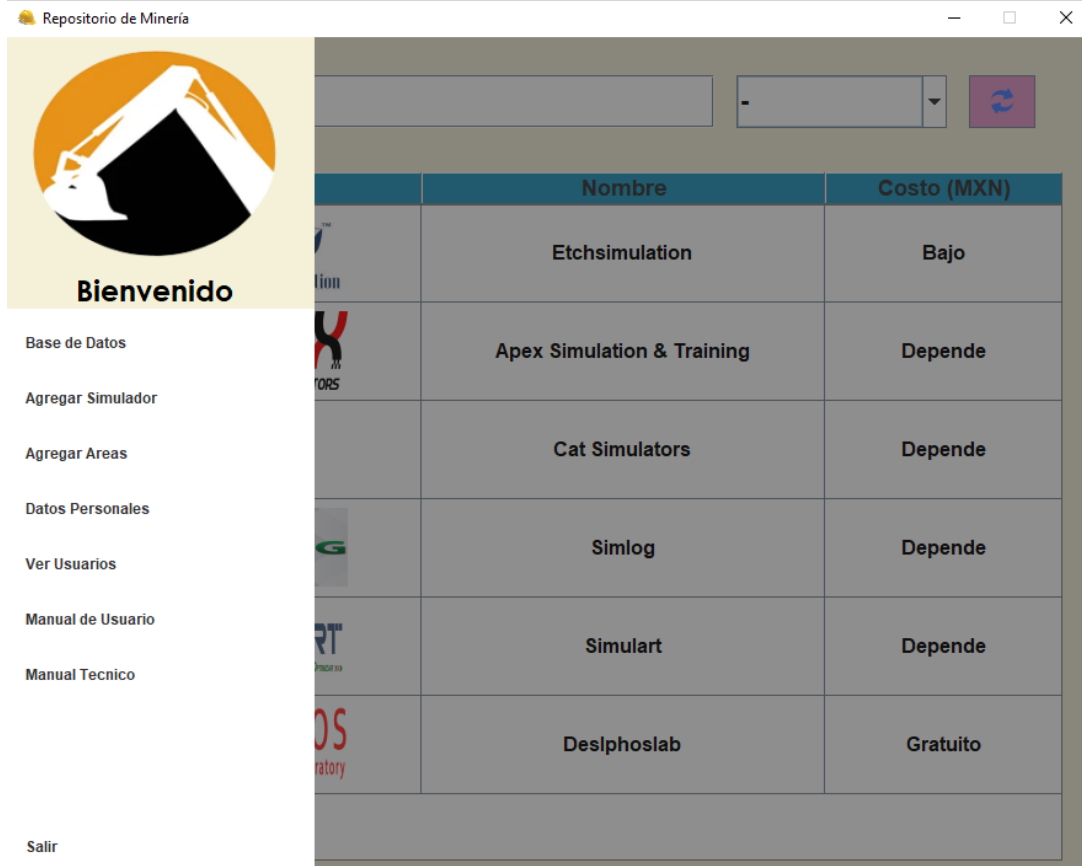
Menu

Logo	Nombre	Costo (MXN)
	Etchsimulation	Bajo
	Apex Simulation & Training	Depende
	Cat Simulators	Depende
	Simlog	Depende
	Simulart	Depende
	Deslphoslab	Gratuito

Esta es la clase donde se maneja la pantalla nexco, aquella que conecta con casi todas las demás pantallas de alguna forma u otra. Las variables y objetos de la clase son los siguientes:

```
private DrawerController drawerController;
private String idUsuario = "";
private String username = "";
private String rolUsuario = "";
```

- **idUsuario**, **username** y **rolUsuario**, variables de tipo String que guardan los datos del usuario que inicia sesión a esa instancia de la pantalla principal.
- **DrawerController**, un objeto que controla el menú lateral izquierdo.



Esta clase también tiene el método **getIconImage** para elegir el ícono que usará esta pantalla:

```
@Override
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit()
        .getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El método constructor de esta clase recibe 3 argumentos: **idUser**, **user**, y **rol**, todas variables de tipo String.

El método inicializa los componentes de la interfaz y checa el rol del usuario. Si el usuario tiene como rol “Usuario”; hará invisibles las opciones para abrir las pantallas AgregarSimuladorPanel, AgregarAreasPanel, VerUsuariosPanel o abrir el manual técnico. Si, por el contrario, el usuario tiene como rol “Admin”, la única pantalla que esconderá será la de

VerUsuarios.

```
if (rolUsuario.equals("Usuario")) {  
    drawerController = Drawer.newDrawer(this)  
        .header(new HeaderPanel())  
        .space(5)  
        .addChild(new DrawerItem("Base de Datos").build())  
        .addChild(new DrawerItem("Datos Personales").build())  
        .addChild(new DrawerItem("Manual de Usuario").build())  
        .addFooter(new DrawerItem("Salir").build())  
        .event(new EventDrawer() {  
            @Override  
            public void selected(int i, DrawerItem di) {  
                switch (i) {  
                    case 0 ->  
                        showBaseDeDatos();  
                    case 1 ->  
                        showDatosPersonales();  
                    case 2 ->  
                        openPDF("manual_usuario.pdf");  
                    case 3 ->  
                        salir();  
                    default -> {  
                    }  
                }  
            }  
        })  
    .build();  
}
```

```

} else if (rolUsuario.equals("Admin")) {
    drawerController = Drawer.newDrawer(this)
        .header(new JLabel("Bienvenido"))
        .space(5)
        .addChild(new DrawerItem("Base de Datos").build())
        .addChild(new DrawerItem("Agregar Simulador").build())
        .addChild(new DrawerItem("Agregar Areas").build())
        .addChild(new DrawerItem("Datos Personales").build())
        .addChild(new DrawerItem("Manual de Usuario").build())
        .addFooter(new DrawerItem("Salir").build())
        .event(new EventDrawer() {
            @Override
            public void selected(int i, DrawerItem di) {
                switch (i) {
                    case 0 ->
                        showBaseDeDatos();
                    case 1 ->
                        showAgregarSimulador();
                    case 2 ->
                        showAgregarAreas();
                    case 3 ->
                        showDatosPersonales();
                    case 4 ->
                        openPDF("manual_usuario.pdf");
                    case 5 ->
                        salir();
                    default -> {
                        }
                }
            }
        })
    .build();
}

```

```

} else if (rolUsuario.equals("MainAdmin")) {
    drawerController = Drawer.newDrawer(this)
        .header(new HeaderPanel())
        .space(5)
        .addChild(new DrawerItem("Base de Datos").build())
        .addChild(new DrawerItem("Agregar Simulador").build())
        .addChild(new DrawerItem("Agregar Areas").build())
        .addChild(new DrawerItem("Datos Personales").build())
        .addChild(new DrawerItem("Ver Usuarios").build())
        .addChild(new DrawerItem("Manual de Usuario").build())
        .addChild(new DrawerItem("Manual Tecnico").build())
        .addFooter(new DrawerItem("Salir").build())
        .event(new EventDrawer() {
            @Override
            public void selected(int i, DrawerItem di) {
                switch (i) {
                    case 0 ->
                        showBaseDeDatos();
                    case 1 ->
                        showAgregarSimulador();
                    case 2 ->
                        showAgregarAreas();
                    case 3 ->
                        showDatosPersonales();
                    case 4 ->
                        showVerUsuarios();
                    case 5 ->
                        openPDF("manual_usuario.pdf");
                    case 6 ->
                        openPDF("manual_tecnico.pdf");
                    case 7 ->
                        salir();
                    default -> {
                        }
                }
            }
        })
    .build();
}

```

Los siguientes métodos se encargan de cargar cada pantalla del programa:  
**DatabaseMenuPanel,**                    **AgregarSimuladorPanel,**                    **AgregarAreasPanel,**  
**DatosPersonalesPanel, VerUsuariosPanel** e **InicioSesionFrame.**

```
private void showBaseDeDatos() {
    DatabaseMenuPanel dbMenuPanel = new DatabaseMenuPanel(this, rolUsuario);
    screensPanel.removeAll();
    screensPanel.add(dbMenuPanel);
    screensPanel.revalidate();
}

private void showAgregarSimulador() {
    AgregarSimuladorPanel agregarSimuladorPanel = new AgregarSimuladorPanel();
    screensPanel.removeAll();
    screensPanel.add(agregarSimuladorPanel);
    screensPanel.revalidate();
}

private void showAgregarAreas() {
    AgregarAreasPanel agregarAreasPanel = new AgregarAreasPanel();
    screensPanel.removeAll();
    screensPanel.add(agregarAreasPanel);
    screensPanel.revalidate();
}

private void showDatosPersonales() {
    DatosPersonalesPanel datosPersonalesPanel = new DatosPersonalesPanel(idUsuario, username, rolUsuario, this);
    screensPanel.removeAll();
    screensPanel.add(datosPersonalesPanel);
    screensPanel.revalidate();
}

private void showVerUsuarios() {
    VerUsuariosPanel verUsuariosPanel = new VerUsuariosPanel(username);
    screensPanel.removeAll();
    screensPanel.add(verUsuariosPanel);
    screensPanel.revalidate();
}

private void salir() {
    this.dispose();
    new InicioSesionFrame().setVisible(true);
}
```

El último método se encarga de abrir los manuales de usuario y técnico.

```
private void openPDF(String filename) {
    try {
        Path tempOutput = null;
        tempOutput = Files.createTempFile(filename, ".pdf");
        tempOutput.toFile().deleteOnExit();
        InputStream is = getClass().getResourceAsStream("/documentacion/" + filename);
        Files.copy(is, tempOutput, StandardCopyOption.REPLACE_EXISTING);
        if (Desktop.isDesktopSupported()) {
            Desktop desktop = Desktop.getDesktop();
            if (desktop.isSupported(Desktop.Action.OPEN)) {
                desktop.open(tempOutput.toFile());
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 8.3.4 DatabaseMenuPanel

En este panel se muestra la base de datos de los simuladores. En el método constructor se encarga de posicionar varios de los elementos de la pantalla, iniciando por **tablaSimuladores**:

```
tablaSimuladores = verTabla();

jPanell.setLayout(new BorderLayout());
jPanell.add(new JScrollPane(tablaSimuladores), BorderLayout.CENTER);
jPanell.revalidate();
jPanell.repaint();
```

Inicializa a **tablaSimuladores** llamando al método **verTabla** de la misma clase (el cual se verá más adelante), y lo posiciona dentro de un JPanel.

Después, a **tablaSimuladores** se le asigna un Mouse Listener, el cual activará un evento cuando se presione una fila de la tabla dos veces. Obtiene el nombre del simulador seleccionando el valor de la columna 1 de la fila seleccionada, y abre una nueva ventana de la clase **VerSimulador**:

```
tablaSimuladores.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        if(e.getClickCount() == 2 && SwingUtilities.isLeftMouseButton(e)) {
            String nombreSimulador = tablaSimuladores.getValueAt(tablaSimuladores.getSelectedRow(), 1)
                .toString();
            new VerSimulador(frame, nombreSimulador, ipAddress).setVisible(true);
        }
    }
});
```

Posteriormente, se especifican las características y posicionamiento de los componentes dentro del menú superior de la pantalla:

```
searchButton.setBackground(new Color(253,193,1));
searchButton.setFont(new Font("Arial", Font.BOLD, 14));
searchButton.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);

searchText.setBackground(new Color(255,255,51));
searchText.setFont(new Font("Arial", Font.BOLD, 14));

searchButton.addActionListener((ActionEvent evt) -> {
    String nombre = searchText.getText().trim();
    nombre = nombre.substring(0, 1).toUpperCase() + nombre.substring(1);
    tableModelSimuladores.setRowCount(0);
    buscarSimulador(nombre);
});
```

A **searchButton** se le asigna un evento cuando es presionado, el cual guarda el valor de **searchText** como un String y busca a el simulador con el nombre que corresponda llamando al método **buscarSimulador** de la misma clase, el cual se verá más adelante.

```
refreshButton.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
refreshButton.setBackground(new Color(153,177,251));
refreshButton.setFont(new Font("Arial", Font.BOLD, 14));

refreshButton.addActionListener((ActionEvent evt) -> {
    tableModelSimuladores.setRowCount(0);
    this.verTabla();
});
```



**refreshButton** también tiene un evento, el cual vacía la tabla actual y vuelve a llamar al método **verTabla** para ver todos los simuladores en la base de datos.

```
menuBar.add(searchButton);
menuBar.add(searchText);

List areas = new List();
areas = new Consultas(ipAddress).getListAreas();

comboBoxAreas.setBackground(new Color(255,255,255));
comboBoxAreas.setFont(new Font("Arial", Font.BOLD, 14));
comboBoxAreas.removeAllItems();
comboBoxAreas.addItem("-");

for(int i = 0; i < areas.getItemCount(); i++){
    comboBoxAreas.addItem(areas.getItem(i));
}
```

**comboBoxAreas** es llenada con los nombres de las áreas en la base de datos, las cuales son previamente guardadas en una lista que llama al método **getListAreas** de la clase **Consultas**.

```
comboBoxAreas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String nombreArea = comboBoxAreas.getSelectedItem().toString();
        if(!nombreArea.equals("-")) {
            tableModelSimuladores.setRowCount(0);
            filtrarPorAreas(nombreArea);
            comboBoxAreas.removeItem("-");
        }
    }
});
```

**comboBoxAreas** también tiene un evento, el cual se activa cuando se selecciona alguno de los elementos de la lista. Llama al método **filtrarPorAreas**, vaciando la tabla actual y creando una nueva tabla sólo con los simuladores que pertenezcan al área seleccionada.

El siguiente método en la clase es **verTabla**. Este método se encarga de recolectar el logo, el

nombre y el costo de cada simulador para plasmarlo en la tabla de la pantalla principal.

```
public JTable verTabla() {
    JTable tabla = new JTable() {
        @Override
        public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
            Component component = super.prepareRenderer(renderer, row, column);
            int rendererWidth = component.getPreferredSize().width;
            TableColumn tableColumn = getColumnModel().getColumn(column);
            tableColumn.setPreferredWidth(Math.max(rendererWidth + getInterCellSpacing().width,
                                                    tableColumn.getPreferredWidth()));
            return component;
        }
    };
}
```

El método empieza creando un objeto llamado “tabla” de la clase JTable, llamando a al método **prepareRenderer**, el cual ayudará a ajustar el ancho cada columna al contenido de las casillas.

```
TablaImagen imgRenderer = new TablaImagen();
imgRenderer.setHorizontalAlignment(JLabel.CENTER);
tabla.setDefaultRenderer(Object.class, imgRenderer);
tabla.setRowHeight(60);
tabla.setDefaultEditor(Object.class, null);
tabla.setFillViewportHeight(true);
tabla.getTableHeader().setReorderingAllowed(false);
tabla.setFont(new Font("Arial", Font.BOLD, 16));
tabla.getTableHeader().setOpaque(false);
tabla.getTableHeader().setBackground(new Color(253,193,1));
tabla.getTableHeader().setFont(new Font("Arial", Font.BOLD, 18));
```

Posteriormente, se definen las características de la tabla. Cada línea significa lo siguiente:

1. Creación de un objeto de la clase **TablaImagen** llamado “imgRenderer”, que permitirá poner JLabels, y por tanto, imágenes en las columnas.
2. Definir la alineación de imgRenderer. En este caso se puso como “CENTER”, para que la imagen esté en el centro.
3. Definir el renderizador de la tabla, en este caso, imgRenderer.
4. Se define la altura de cada fila (60 px).
5. Se define el editor de la tabla como *null* para evitar que los usuarios modifiquen los

- valores de cada columna al dar doble clic.
6. Se establece que la columna llenará el alto de cada fila.
  7. Se impide que el usuario pueda alterar el orden de las columnas con el mouse.
  8. Se define la tipografía de la tabla.
  9. Se define que los encabezados de la tabla no serán opacos, para permitir cambiar sus colores.
  10. Se define el color de los encabezados.
  11. Se define la tipografía de los encabezados.

Posteriormente, se conecta a la base de datos y se recolectan el logo, nombre y costo de cada simulador y se agregan como una fila a **tableModelSimuladores**.

```
try{
    Connection cn = new Conexion(ipAddress).conectar();
    PreparedStatement pst = cn.prepareStatement(consulta);
    ResultSet rs = pst.executeQuery();

    while(rs.next()){
        Object[] fila = new Object[4];

        Image dimg = null;
        try{
            BufferedImage im = ImageIO.read(rs.getBinaryStream("logo"));
            dimg = im.getScaledInstance(110, 64, Image.SCALE_SMOOTH);
            ImageIcon icon = new ImageIcon(dimg);
            fila[0] = new JLabel(icon);
        }
        catch(Exception e){
            fila[0] = "";
        }

        fila[1] = rs.getString("nombre_simulador");

        if(rs.getString("costo").equals("") || rs.getString("costo").equals("0"))
            fila[2] = "GRATIS";
        else
            fila[2] = rs.getString("costo");

        tableModelSimuladores.addRow(fila);
    }
}
```

Una vez termina de recolectarse toda la información, se define el modelo de la tabla y el método regresa al objeto “tabla”.

```
        tabla.setModel(tableModelSimuladores);  
        cn.close();  
        pst.close();  
        rs.close();  
    } catch (SQLException e) {  
        e.printStackTrace();  
        System.out.println(e.getErrorCode());  
    }  
  
    return tabla;  
}
```

Los siguientes dos métodos, **buscarSimulador** y **filtrarPorAreas** son básicamente idénticos a **verTabla**, con la diferencia de que reciben atributos y la consulta que realizan a la base de datos.

**buscarSimulador** recibe un String, y posteriormente buscará en la base de datos si hay un simulador con ese nombre.

```
String consulta = "SELECT logo, nombre_simulador, costo "  
+ "FROM simuladores "  
+ "WHERE nombre_simulador = '" + nombre + "' "  
+ "GROUP BY simuladores.id_simulador";
```

**filtrarPorAreas** también recibe un String, pero este buscará en la base de datos por todos los simuladores que pertenezcan a esa área usando el ID del área, el cual obtendrá llamando al método *getIDArea* de la clase **Areas**.

```
String id_area = new Areas(ipAddress).getIDArea(area);

String consulta = "SELECT logo, costo, nombre_simulador "
+ "FROM simuladores, areas, simuladorarea "
+ "WHERE simuladores.id_simulador = simuladorarea.id_simulador "
+ "AND simuladorarea.id_area = areas.id_area "
+ "AND simuladorarea.id_area = '" + id_area + "'";
```

### 8.3.5. AgregarSimuladorPanel

Repositorio de Minería

**Los campos en color azul son obligatorios**

Menu

Logo

Nombre del simulador:

Area:

Requerimientos:

Tutorial:

Costo:

Características:

Link:

Fecha de consulta:

La clase tiene las siguientes variables y objetos:

```
private Simuladores simuladores;
private Areas objetoAreas;

private final JFileChooser fileChooser = new JFileChooser();
private FileNameExtensionFilter imageFilter = new FileNameExtensionFilter("Image files", ImageIO.getReaderFileSuffixes());
private File file = null;

private JDateChooser dateChooser = new JDateChooser("yyyy/MM/dd", "###/##/##", '_');
private JTextFieldDateEditor textFieldDateEditor = (JTextFieldDateEditor) dateChooser.getDateEditor();

private JComboBox comboBoxAreas = new JComboBox();
```

- **simuladores**, objeto de la clase Simuladores.
- **objetoAreas**, objeto de la clase Areas.
- **fileChooser**, objeto de la clase JFileChooser, y que será la parte gráfica que nos permitirá elegir una imagen de nuestra computadora.
- **imageFilter**, objeto de la clase FileNameExtensionFilter, que servirá como filtro para sólo poder elegir imágenes a través de **fc**.
- **file**, objeto de la clase File, que servirá para obtener los detalles de la imagen que escojamos.
- **dateChooser**, objeto de la clase JDateChooser, elemento gráfico que nos permitirá escoger una fecha usando un calendario.
- **textFieldDateEditor**, objeto de la clase JTextFieldDateEditor, que será el campo de texto que mostrará la fecha que seleccionemos.
- **comboBoxAreas**, objeto de la clase JComboBox, que mostrará las áreas en la base de datos.

Esta clase también tiene el método **getIconImage** para elegir el ícono que usará esta pantalla:

```
@Override
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit()
        .getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El método constructor no hace muchas cosas más allá de inicializar los componentes gráficos de la interfaz.

```
public AgregarSimuladorPanel() {
    initComponents();

    this.objetoAreas = new Areas();
    this.simuladores = new Simuladores();

    comboTutorial.removeAllItems();
    comboTutorial.addItem("Si");
    comboTutorial.addItem("No");

    comboBoxAreas = new Consultas().getAreas();
    comboBoxAreas.setFont(new Font("Century Gothic", Font.BOLD, 18));
    comboBoxAreas.setBounds(340, 170, 150, 25);
    jLabelFondo.add(comboBoxAreas);

    textReq.setLineWrap(true);
    textReq.setWrapStyleWord(true);

    textCaracteristicas.setLineWrap(true);
    textCaracteristicas.setWrapStyleWord(true);

    dateChooser.setBounds(340, 590, 250, 22);
    dateChooser.setFont(new Font("Century Gothic", Font.BOLD, 18));
    jLabelFondo.add(dateChooser);
    textFieldDateEditor.setEditable(false);
}
```

El siguiente método es **botonAbrirActionPerformed**, el cual se llama cuando se presiona el botón “Abrir imagen”.

```
private void botonAbrirActionPerformed(java.awt.event.ActionEvent evt) {
    fc.setFileFilter(imageFilter);
    try{
        int returnVal = fc.showOpenDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            botonAbrir.setText(file.getName());
        }
    }
    catch (HeadlessException e) {
        e.printStackTrace();
    }
}
```

Esto lo que hace es abrir el JFileChooser, y una vez que se elige una imagen, se guarda en el objeto **file**, y el botón pasa a tener el texto del nombre del archivo.



El siguiente método es **botonAgregarActionPerformed**, el cual se llama cuando se presiona el botón “Agregar”. El método lo que hace es lo siguiente:

1. Guardar los datos ingresados en variables y objetos.
2. Si **file** no es nulo (es decir, si se escogió una imagen), se guardan los datos de la imagen en un objeto de tipo `InputStream` llamado **fis**.

```
try{
    File image;
    InputStream fis = null;
    if(file != null){
        image = new File(file.getAbsolutePath());
        fis = new FileInputStream(image);
    }
    String nombre = textNombre.getText().trim();
    String area = areas.getSelectedItemAt().toString();
    String requerimientos = textReq.getText();
    String tutorial = comboTutorial.getSelectedItemAt().toString();
    String costo = textCosto.getText().trim();
    String características = textCaracterísticas.getText();
    String link = textLink.getText().trim();
    java.sql.Date fechaSQL = new java.sql.Date(calendar.getDate().getTime());
    String fecha = fechaSQL.toString();
}
```

3. Se verifica que los campos “nombre” y “área” no estén vacíos.
4. Se hace la consulta.

```
if(!nombre.isEmpty() && !area.isEmpty()){
    try{
        String checkSimuladores = "SELECT * FROM simuladores WHERE nombre_simulador = '" + nombre + "'";
        Connection cn = new Conexion(ipAddress).conectar();
        PreparedStatement pstSimuladores = cn.prepareStatement(insertarSimuladores);
        PreparedStatement pstCheckSimuladores = cn.prepareStatement(checkSimuladores);

        ResultSet rsCheckSimuladores = pstCheckSimuladores.executeQuery();
    }
}
```

5. Se verifica que no exista una entrada en la base de datos con el mismo nombre. De ser así, se continúa con la operación.
6. Si **file** no es nulo, se introduce **fis** a la base de datos. Caso contrario, se ingresa la imagen por defecto llamada “no\_image.jpg”.



```
if(file != null)
    pstSimuladores.setBinaryStream(2, fis, (int) file.length());
else
    pstSimuladores.setBinaryStream(2, RepositorioMineria.class.getResourceAsStream
        ("/imagenes/no_image.jpg"));
```

7. Se insertan todos los datos en la base de datos.

```
if(!rsCheckSimuladores.next()){
    pstSimuladores.setString(1, "0");

    if(file != null)
        pstSimuladores.setBinaryStream(2, fis, (int) file.length());
    else
        pstSimuladores.setBinaryStream(2, RepositorioMineria.class.getResourceAsStream
            ("/imagenes/no_image.jpg"));

    pstSimuladores.setString(3, nombre);
    pstSimuladores.setString(4, requerimientos);
    pstSimuladores.setString(5, tutorial);
    pstSimuladores.setString(6, costo);
    pstSimuladores.setString(7, características);
    pstSimuladores.setString(8, link);
    pstSimuladores.setString(9, fecha);

    pstSimuladores.executeUpdate();

    JOptionPane.showMessageDialog(this, "Simulador agregado.");

    insertarArea(nombre);

    DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
    model.setRowCount(0);
    pp.verTabla();

    botonAbrir.setText("Abrir imagen");
    textNombre.setText("");
    textReq.setText("");
    textCosto.setText("");
    textCaracteristicas.setText("");
    textLink.setText("");
```

El último método en esta clase es **insertarArea**, el cual recibe el nombre del área seleccionada y lo inserta en la tabla simuladorarea.

```
private void insertarArea(String nombre) {  
    try{  
        Connection cn = new Conexion(ipAddress).conectar();  
        String n = nombre;  
        String area = areas.getSelectedItem().toString();  
        String id_simulador = sim.getIDSimulador(n);  
        String id_area = objetoAreas.getIDArea(area);  
  
        PreparedStatement pstInsertarArea = cn.prepareStatement("insert into Simuladorarea values(?,?)");  
  
        pstInsertarArea.setString(1, id_simulador);  
        pstInsertarArea.setString(2, id_area);  
  
        pstInsertarArea.executeUpdate();  
  
        System.out.println("Insercion en simuladorarea exitosa");  
  
        cn.close();  
        pstInsertarArea.close();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
}
```

### 8.3.6. VerSimuladorFrame

Esta clase tiene los siguientes objetos y variables:

- **myFrame**, objeto de la clase MainScreenFrame.
- **myPanel**, objeto de clase DatabaseMenuPanel.
- **buttonGroup**, objeto de la clase ButtonGroup.
- **fileChooser**, objeto de la clase JFileChooser, y que será la parte gráfica que nos permitirá elegir una imagen de nuestra computadora.
- **imageFilter**, objeto de la clase FileNameExtensionFilter, que servirá como filtro para sólo poder elegir imágenes a través de **fc**.
- **file**, objeto de la clase File, que servirá para obtener los detalles de la imagen que escogamos.
- **id\_simulador**, variable de tipo String.
- **areas\_actuales**, arreglo de 5 Strings.
- **dateChooser**, objeto de la clase JDateChooser, elemento gráfico que nos permitirá

escoger una fecha usando un calendario.

- **textEditor**, objeto de la clase `JTextFieldDateEditor`, que será el campo de texto que mostrará la fecha que seleccionemos.
- **nombreViejo**, variable de tipo `String`.
- **rolUsuario**, variable de tipo `String`.

Esta clase también tiene el método **getIconImage**, que escoge y asigna el ícono de la pantalla:

```
@Override
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit().getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El método constructor recibe 4 argumentos: un objeto de la clase `MainScreenFrame`, un objeto de la clase `DatabaseMenuPanel`, un `String` que será el nombre del simulador seleccionado en la tabla de la pantalla principal, y un `String` que será el rol del usuario.

El siguiente método es **displayData**, el cual se encarga de recopilar todos los datos del simulador usando su ID y mostrarlos en la interfaz.

```
public void displayData() {  
    String consulta = "SELECT simuladores.* "  
        + "FROM simuladores "  
        + "WHERE simuladores.id_simulador = '" + id_simulador + "' ";  
  
    try{  
        Connection cn = new Conexion(ipAddress).conectar();  
        PreparedStatement pstConsulta = cn.prepareStatement(consulta);  
        ResultSet rs = pstConsulta.executeQuery();  
        while(rs.next()) {  
            textNombre.setText(rs.getString("nombre_simulador"));  
  
            try{  
                BufferedImage fis = ImageIO.read(rs.getBinaryStream("logo"));  
                Image dimg = fis.getScaledInstance(labelLogo.getWidth(),  
                    labelLogo.getHeight(), Image.SCALE_SMOOTH);  
                labelLogo.setIcon(new ImageIcon(dimg));  
            }  
            catch(Exception e) {  
                e.printStackTrace();  
            }  
  
            textRequerimientos.setText(rs.getString("requerimientos"));  
  
            String tutorial = rs.getString("tutorial");  
            group.clearSelection();  
            if(tutorial.equals("Si"))  
                radioSi.setSelected(true);  
            else if(tutorial.equals("No"))  
                radioNo.setSelected(true);  
        }  
    }  
}
```

```
String costo = rs.getString("costo");

if(costo.equals("") || costo.equals("0"))
    textCosto.setText("GRATIS");
else
    textCosto.setText(rs.getString("costo"));

displayAreas();

textCaracteristicas.setText(rs.getString("caracteristicas"));

textLink.setText(rs.getString("link"));

calendar.setDate(rs.getDate("fecha_consulta"));
}

cn.close();
pstConsulta.close();
rs.close();
}
catch(Exception e){
    e.printStackTrace();
}
}
```

El siguiente método es **displayAreas**, el cual se encarga de llenar los JComboBoxes y al arreglo *areas\_actuales* con las áreas a las cuales pertenece el simulador seleccionado.

```
public void displayAreas() {
    String consultaAreas = "SELECT * "
        + "FROM simuladorarea, areas "
        + "WHERE simuladorarea.id_simulador = '" + id_simulador + "' "
        + "AND simuladorarea.id_area = areas.id_area";

    try{
        Connection cn = new Conexion(ipAddress).conectar();
        PreparedStatement pstConsultaArea = cn.prepareStatement(consultaAreas);
        ResultSet rsArea = pstConsultaArea.executeQuery();
        int i = 1;
        while(rsArea.next()) {
            switch(i) {
                case 1: comboBoxArea1.setSelectedItem(rsArea.getString("nombre_area"));
                    areas_actuales[0] = comboBoxArea1.getSelectedItem().toString();
                    break;
                case 2: comboBoxArea2.setSelectedItem(rsArea.getString("nombre_area"));
                    areas_actuales[1] = comboBoxArea2.getSelectedItem().toString();
                    break;
                case 3: comboBoxArea3.setSelectedItem(rsArea.getString("nombre_area"));
                    areas_actuales[2] = comboBoxArea3.getSelectedItem().toString();
                    break;
                case 4: comboBoxArea4.setSelectedItem(rsArea.getString("nombre_area"));
                    areas_actuales[3] = comboBoxArea4.getSelectedItem().toString();
                    break;
                case 5: comboBoxArea5.setSelectedItem(rsArea.getString("nombre_area"));
                    areas_actuales[4] = comboBoxArea5.getSelectedItem().toString();
                    break;
                default: break;
            }
            i++;
        }

        cn.close();
        pstConsultaArea.close();
        rsArea.close();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

Al igual que en la clase AgregarSimulador, la clase VerSimulador tiene un método para seleccionar una imagen, llamada **botonImageActionPerformed**:

```
private void botonImagenActionPerformed(java.awt.event.ActionEvent evt) {
    fc.setFileFilter(imageFilter);
    fc.setAcceptAllFileFilterUsed(false);
    try{
        int returnVal = fc.showOpenDialog(this);
        if(returnVal == JFileChooser.APPROVE_OPTION){
            file = fc.getSelectedFile();
            botonImagen.setText(file.getName());
        }
    }
    catch(HeadlessException e){
        e.printStackTrace();
    }
}
```

El método **botonActualizarActionPerformed** se llama al presionar el botón “Actualizar”. Las únicas restricciones de este método es que el ni el nombre ni las áreas pueden estar vacíos y que el nuevo nombre del simulador no puede ser igual a alguno de los que ya existen en la base de datos.

```
private void botonActualizarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        File image;
        InputStream fis = new Consultas(ipAddress).getLogo(id_simulador);
        if(file != null){
            image = new File(file.getAbsolutePath());
            fis = new FileInputStream(image);
        }
        else{

        }

        String nombre = textNombre.getText().trim();

        String requerimientos = textRequerimientos.getText().trim();

        String tutorial = "";
        if(radioSi.isSelected())
            tutorial = "Si";
        else
            tutorial = "No";

        String costo = textCosto.getText().trim();
```



```

ArrayList<String> areas = new ArrayList<>();
areas.add(comboBoxArea1.getSelectedItem().toString());
areas.add(comboBoxArea2.getSelectedItem().toString());
areas.add(comboBoxArea3.getSelectedItem().toString());
areas.add(comboBoxArea4.getSelectedItem().toString());
areas.add(comboBoxArea5.getSelectedItem().toString());

String caracteristicas = textCaracteristicas.getText().trim();

String link = textLink.getText().trim();

java.sql.Date fechaSQL = new java.sql.Date(calendar.getDate().getTime());
String fecha = fechaSQL.toString();

String update = "UPDATE simuladores "
                + "SET logo = ?, nombre_simulador = ?, requerimientos = ?, "
                + "tutorial = ?, costo = ?, caracteristicas = ?, link = ?, "
                + "fecha_consulta = ? "
                + "WHERE id_simulador = '" + id_simulador + "'";

String checkSimuladores = "SELECT * FROM simuladores WHERE nombre_simulador = '" + nombre + "'";

try{
    Connection cn = new Conexion(ipAddress).conectar();
    PreparedStatement pstUpdate = cn.prepareStatement(update);
    PreparedStatement pstCheckSimuladores = cn.prepareStatement(checkSimuladores);

    ResultSet rsCheckSimuladores = pstCheckSimuladores.executeQuery();

    if(!rsCheckSimuladores.next() || nombreViejo.equals(nombre)){
        if(file != null)
            pstUpdate.setBinaryStream(1, fis, (int) file.length());
    }
}

```

```

else
    pstUpdate.setBinaryStream(1, fis);

    pstUpdate.setString(2, nombre);
    pstUpdate.setString(3, requerimientos);
    pstUpdate.setString(4, tutorial);
    pstUpdate.setString(5, costo);
    pstUpdate.setString(6, características);
    pstUpdate.setString(7, link);
    pstUpdate.setString(8, fecha);

    if(!nombre.isEmpty() && !areas.isEmpty()){
        pstUpdate.executeUpdate();
        actualizarAreas(areas);
        DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
        model.setRowCount(0);
        pp.verTabla();
        JOptionPane.showMessageDialog(null, "Actualizacion exitosa");
        nombreViejo = nombre;
    }
    else
        JOptionPane.showMessageDialog(null, "Faltan llenar campos importantes");

    cn.close();
    pstUpdate.close();
}
else{
    JOptionPane.showMessageDialog(null, "Ya existe un simulador con ese nombre");
    rsCheckSimuladores.close();
}
}

catch(Exception e){
    e.printStackTrace();
}

catch(Exception e){
    e.printStackTrace();
}
}

```

El método **actualizarAreas** recibe un `ArrayList` desde el método **botonActualizarActionPerformed**, y sirve para actualizar la tabla *simuladorarea* en la base de datos, modificando las áreas a las que pertenece un simulador.

```
public void actualizarAreas(ArrayList areas){
    try{
        String updateAreas = "UPDATE simuladorarea "
                               + "SET id_area = ? "
                               + "WHERE id_simulador = '" + id_simulador + "' "
                               + "AND id_area = ?";

        String insertAreas = "INSERT INTO simuladorarea values(?,?)";

        String check = "SELECT simuladorarea.* "
                       + "FROM simuladorarea "
                       + "WHERE simuladorarea.id_simulador = '" + id_simulador + "' "
                       + "AND simuladorarea.id_area = ?";

        Connection cn = new Conexion(ipAddress).conectar();

        PreparedStatement pstUpdateAreas = cn.prepareStatement(updateAreas);
        PreparedStatement pstInsertAreas = cn.prepareStatement(insertAreas);
        PreparedStatement pstCheck = cn.prepareStatement(check);

        ResultSet rsCheck;

        for(int i = 0; i < areas.size(); i++){
            String idArea = "";
            if(areas.get(i).toString().equals("-") && !areas.get(i).equals(areas_actuales[i])){
                idArea = new Areas(ipAddress).getIDArea(areas_actuales[i]);
                eliminarArea(idArea);
                areas_actuales[i] = areas.get(i).toString();
            }
            else{
                idArea = new Areas(ipAddress).getIDArea(areas.get(i).toString());
                pstCheck.setString(1, idArea);

                rsCheck = pstCheck.executeQuery();

                if(rsCheck.next()){
                    //No hagas nada, ya existe ese registro
                }
                else if(areas_actuales[i].equals("-") && !areas.get(i).equals(areas_actuales[i])){
                    pstInsertAreas.setString(1, id_simulador);
                    pstInsertAreas.setString(2, idArea);
                    pstInsertAreas.executeUpdate();
                    areas_actuales[i] = areas.get(i).toString();
                }
                else{
                    String idAreaOriginal = new Areas(ipAddress).getIDArea(areas_actuales[i]);
                    pstUpdateAreas.setString(1, idArea);
                    pstUpdateAreas.setString(2, idAreaOriginal);
                    pstUpdateAreas.executeUpdate();
                    areas_actuales[i] = areas.get(i).toString();
                }
            }
        }
    }
}
```

Dentro del ciclo for se recorre el ArrayList que se recibió y se hace lo siguiente:

1. Se inicializa una variable de tipo String donde se guardará el id del área.

2. Se checa si el valor del área seleccionada en el ArrayList es igual a “-” y es diferente al valor actual de *areas\_actuales*. De ser así, se llama al método **eliminarArea**, el cual se describirá más adelante.
3. Si la evaluación anterior da falso, se procede a actualizar los valores en *simuladorarea*.
4. Se checa si el valor en *areas\_actuales* es igual a “-” y el valor en el ArrayList es diferente a *areas\_actuales*. De ser así, se inserta el valor del ArrayList en la tabla *simuladorarea*.
5. Caso contrario, se actualiza la tabla *simuladorarea*.

En el método **eliminarArea**, se elimina el área en la tabla *simuladorarea* que contenga el ID que le mande el método **actualizarArea** y el ID del simulador que se acaba de actualizar.

```
private void eliminarArea(String id_area){
    try{
        String delete = "DELETE FROM simuladorarea "
            + "WHERE id_simulador = '" + id_simulador + "' "
            + "AND id_area = '" + id_area + "'";

        String check = "SELECT * "
            + "FROM simuladorarea "
            + "WHERE id_simulador = '" + id_simulador + "' "
            + "AND id_area = '" + id_area + "'";

        Connection cn = new Conexion(ipAddress).conectar();
        PreparedStatement pstDelete = cn.prepareStatement(delete);
        PreparedStatement pstCheck = cn.prepareStatement(check);
        ResultSet rsCheck = pstCheck.executeQuery();

        int yes_no = JOptionPane.showConfirmDialog(null,
            "¿Seguro que quiere eliminar una o más áreas de este simulador?", "Alerta",
            JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE);

        if (yes_no == 0) {
            if (rsCheck.next()) {
                pstDelete.executeUpdate();
                DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
                model.setRowCount(0);
                pp.verTabla();
            }
            JOptionPane.showMessageDialog(null, "Área(s) desasignada(s) exitosamente.");
        }

        cn.close();
        pstDelete.close();
    }
}
```

```

        pstCheck.close();
        rsCheck.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

```

El método **botonEliminarActionPerformed** se activa al presionar el botón “Eliminar”. Borra el simulador seleccionado de la base de datos.

```

private void botonEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String checkSimulador = "SELECT * FROM simuladores WHERE id_simulador = '" + id_simulador + "'";
        String deleteSimulador = "DELETE FROM simuladores WHERE id_simulador = '" + id_simulador + "'";

        Connection cn = new Conexion(ipAddress).conectar();
        PreparedStatement pstDeleteSimulador = cn.prepareStatement(deleteSimulador);
        PreparedStatement pstCheck = cn.prepareStatement(checkSimulador);

        ResultSet rs = pstCheck.executeQuery();

        if(rs.next()){
            int yes_no = JOptionPane.showConfirmDialog(null,
                "¿Seguro que quiere eliminar este simulador?", "Alerta", JOptionPane.YES_NO_OPTION,
                JOptionPane.WARNING_MESSAGE);

            if (yes_no == 0){
                eliminarAreasSimuladores();
                pstDeleteSimulador.executeUpdate();
                JOptionPane.showMessageDialog(null, "Simulador eliminado exitosamente.");
                DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
                model.setRowCount(0);
                pp.verTabla();
                pp.setEnabled(true);
                this.dispose();
            }
        }

        cn.close();
        pstDeleteSimulador.close();
        rs.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

```

El método **eliminarAreasSimuladores** es un método que se llama desde **botonEliminarActionPerformed**, y se encarga de eliminar todos los registros del simulador que se quiere eliminar de la tabla *simuladorarea*.

```
private void eliminarAreasSimuladpres () {  
    try{  
        String checkSimuladorArea = "SELECT * FROM simuladorarea WHERE id_simulador = '" + id_simulador + "'";  
        String deleteSimuladorArea = "DELETE FROM simuladorarea WHERE id_simulador = '" + id_simulador + "'";  
  
        Connection cn = new Conexion(ipAddress).conectar();  
        PreparedStatement pstDeleteSimuladorArea = cn.prepareStatement(deleteSimuladorArea);  
        PreparedStatement pstCheck = cn.prepareStatement(checkSimuladorArea);  
  
        ResultSet rs = pstCheck.executeQuery();  
  
        if(rs.next()){  
            pstDeleteSimuladorArea.executeUpdate();  
        }  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
}
```

### 8.3.7. AgregarArea

Repositorio de Minería

Menu

Areas
Ambientales
Autotransporte
Construcción
Forestal
Maquinaria
Minería
Motorport
Pro Model
Procesos
Simulación

Área seleccionada

Actualizar

Eliminar

Nueva Área

Agregar

En esta clase se tienen los siguientes objetos y variables:

```
private String headerAreas[] = {"Areas"};
private DefaultTableModel tableModelAreas = new DefaultTableModel(headerAreas, 0);
private JTable tablaAreas;
private String nombreAntiguo = "";
```

- **headerAreas**, arreglo de un solo String que tiene el encabezado de la única columna de la tabla.
- **tableModelAreas**, objeto de la clase DefaultTableModel que recibe el arreglo headerAreas como argumento para definir el número de columnas.

- **tablaAreas**, objeto de la clase JTable.
- **nombreAntiguo**, variable de tipo String.

Dentro del método constructor, se define un Mouse Listener para **tablaAreas**, el cual guarda el nombre del área seleccionada en la variable **nombreAntiguo** y lo despliega en el campo de texto **textAreaSeleccionada**.

También posee el método **getIconImage** para seleccionar y mostrar el ícono de la ventana.

```
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit()
        .getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El siguiente método en la clase es **verTablaAreas**. Este método se encarga de recolectar el nombre de cada área para plasmarlo en la tabla de la interfaz.

```
public JTable verTablaAreas() {
    JTable tabla = new JTable() {
        @Override
        public Component prepareRenderer(TableCellRenderer renderer, int row, int column) {
            Component component = super.prepareRenderer(renderer, row, column);
            int rendererWidth = component.getPreferredSize().width;
            TableColumn tableColumn = getColumnModel().getColumn(column);
            tableColumn.setPreferredWidth(Math.max(rendererWidth + getInterCellSpacing().width,
                tableColumn.getPreferredWidth()));
            return component;
        }
    };
}
```

El método empieza creando un objeto llamado “tabla” de la clase JTable, llamando a al método **prepareRenderer**, el cual ayudará a ajustar el ancho cada columna al contenido de las casillas.



```
TablaImagen imgRenderer = new TablaImagen();  
imgRenderer.setHorizontalAlignment(JLabel.CENTER);  
tabla.setDefaultRenderer(Object.class, imgRenderer);  
tabla.setRowHeight(60);  
tabla.setDefaultEditor(Object.class, null);  
tabla.setFillViewportHeight(true);  
tabla.getTableHeader().setReorderingAllowed(false);  
tabla.setFont(new Font("Arial", Font.BOLD, 16));  
tabla.getTableHeader().setOpaque(false);  
tabla.getTableHeader().setBackground(new Color(253,193,1));  
tabla.getTableHeader().setFont(new Font("Arial", Font.BOLD, 18));
```

Posteriormente, se definen las características de la tabla. Cada línea significa lo siguiente:

1. Creación de un objeto de la clase **TablaImagen** llamado “imgRenderer”.
2. Definir la alineación de imgRenderer. En este caso se puso como “CENTER”, para que los contenidos de cada celda estén centrados.
3. Definir el renderizador de la tabla, en este caso, imgRenderer.
4. Se define la altura de cada fila (60 px).
5. Se define el editor de la tabla como *null* para evitar que los usuarios modifiquen los valores de cada columna al dar doble clic.
6. Se establece que la columna llenará el alto de cada fila.
7. Se impide que el usuario pueda alterar el orden de las columnas con el mouse.
8. Se define la tipografía de la tabla.
9. Se define que los encabezados de la tabla no serán opacos, para permitir cambiar sus colores.
10. Se define el color de los encabezados.
11. Se define la tipografía de los encabezados.

Posteriormente, se conecta a la base de datos y se recolectan el nombre cada área y se agregan como una fila a **tableModelAreas**.

```
String consulta = "SELECT nombre_area FROM areas";

try{
    Connection cn = new Conexion(ipAddress).conectar();
    PreparedStatement pst = cn.prepareStatement(consulta);
    ResultSet rs = pst.executeQuery();

    while(rs.next()){
        Object fila[] = new Object[1];

        fila[0] = rs.getString("nombre_area");

        tableModelAreas.addRow(fila);
    }
}
```

Finalmente, se define el modelo de la tabla y se devuelve la tabla resultante.

```
        tabla.setModel(tableModelAreas);
        cn.close();
        pst.close();
        rs.close();
    }
    catch(Exception e){
        e.printStackTrace();
    }

    return tabla;
}
```

El método **botonAgregarActionPerformed** se llama cuando se presiona el botón “Agregar”. Si se dejó la caja de texto en blanco, saldrá una advertencia; caso contrario, se agregará el área a la base de datos.

```
private void botonAgregarActionPerformed(java.awt.event.ActionEvent evt) {
    String nombreArea = textNuevaArea.getText().trim();

    if(!nombreArea.isEmpty()){
        try{
            String check = "SELECT * from areas WHERE nombre_area = '" + nombreArea + "'";
            String insertArea = "INSERT INTO areas values(?,?)";

            Connection cn = new Conexion(ipAddress).conectar();
            PreparedStatement pstCheck = cn.prepareStatement(check);
            PreparedStatement pstInsertArea = cn.prepareStatement(insertArea);

            ResultSet rsCheck = pstCheck.executeQuery();

            if(!rsCheck.next()){
                pstInsertArea.setString(1, "0");
                pstInsertArea.setString(2, nombreArea);
                pstInsertArea.executeUpdate();

                tableModelAreas.setRowCount(0);
                verTablaAreas();

                JOptionPane.showMessageDialog(this, "Área añadida.");

                pp.comboBoxAreas.addItem(nombreArea);

                textNuevaArea.setText("");

                cn.close();
                pstCheck.close();
                pstInsertArea.close();
            }

            else{
                JOptionPane.showMessageDialog(this, "Ya existe esa área.");
            }
            rsCheck.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
    else{
        JOptionPane.showMessageDialog(this, "No puede agregar un valor vacío.");
    }
}
```

El método **botonActualizarActionPerformed** se llama cuando se presiona el botón “Actualizar”. Si no se selecciona alguna de las áreas en la tabla, saldrá una advertencia. De lo contrario, se actualizará el nombre del área seleccionada.

```
private void botonActualizarAreaActionPerformed(java.awt.event.ActionEvent evt) {
    String nombreArea = textAreaSeleccionada.getText().trim();
    if(!nombreArea.isEmpty() && !nombreAntiguo.isEmpty()){
        try {
            String check = "SELECT * from areas WHERE BINARY nombre_area = '" + nombreArea + "'";
            String consulta = "UPDATE areas SET nombre_area = '" + nombreArea + "' "
                + "WHERE nombre_area = '" + nombreAntiguo + "'";
            Connection cn = new Conexion(ipAddress).conectar();
            PreparedStatement pstUpdate = cn.prepareStatement(consulta);
            PreparedStatement pstCheck = cn.prepareStatement(check);

            ResultSet rsCheck = pstCheck.executeQuery();

            if(!rsCheck.next()){
                int yes_no = JOptionPane.showConfirmDialog(null,
                    "¿Seguro que quiere cambiar el nombre de esa área?", "Alerta", JOptionPane.YES_NO_OPTION,
                    JOptionPane.WARNING_MESSAGE);

                if(yes_no == 0){
                    pstUpdate.executeUpdate();
                    JOptionPane.showMessageDialog(null, "Área actualizada correctamente.");
                    tableModelAreas.setRowCount(0);
                    verTablaAreas();
                    DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
                    model.setRowCount(0);
                    pp.verTabla();
                    pp.comboBoxAreas.removeItem(nombreAntiguo);
                    pp.comboBoxAreas.addItem(nombreArea);

                    textAreaSeleccionada.setText("");
                    nombreAntiguo = "";

                    cn.close();
                    pstUpdate.close();
                    pstCheck.close();
                    rsCheck.close();
                }
            }
            else{
                JOptionPane.showMessageDialog(this, "Ya existe esa área.");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    else{
        JOptionPane.showMessageDialog(this, "Debe seleccionar alguno de los simuladores en la lista.");
    }
}
```

El método **botonEliminarActionPerformed** se llama cuando se presiona el botón “Eliminar”. Si no se selecciona alguna de las áreas en la tabla, saldrá una advertencia. De lo contrario, se eliminará el área seleccionada.

```
private void botonEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    String nombreArea = textAreaSeleccionada.getText().trim();
    if(!nombreArea.isEmpty() && !nombreAntiguo.isEmpty()){
        try {
            String id_area = new Areas(ipAddress).getIDArea(nombreArea);
            String delete = "DELETE FROM areas WHERE nombre_area = '" + nombreArea + "'";
            Connection cn = new Conexion(ipAddress).conectar();
            PreparedStatement pstDelete = cn.prepareStatement(delete);

            int yes_no = JOptionPane.showConfirmDialog(null,
                "¿Seguro que quiere eliminar esta área?", "Alerta", JOptionPane.YES_NO_OPTION,
                JOptionPane.WARNING_MESSAGE);

            if (yes_no == 0) {
                eliminarSimuladorArea(id_area);
                pstDelete.executeUpdate();
                JOptionPane.showMessageDialog(null, "Área eliminada correctamente.");
                tableModelAreas.setRowCount(0);
                verTablaAreas();
                DefaultTableModel model = (DefaultTableModel) pp.tablaSimuladores.getModel();
                model.setRowCount(0);
                pp.verTabla();
                pp.comboBoxAreas.removeItem(nombreAntiguo);

                textAreaSeleccionada.setText("");
                nombreAntiguo = "";

                cn.close();
                pstDelete.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    else{
        JOptionPane.showMessageDialog(this, "Debe seleccionar alguno de los simuladores en la lista.");
    }
}
```

El método **eliminarSimuladorArea** se llama desde el método *botonEliminarActionPerformed*, y se encarga de eliminar todas las instancias donde aparece el área seleccionada de la tabla *simuladorarea* en la base de datos.

```
private void eliminarSimuladorArea(String id_area){  
    try{  
        String delete = "DELETE FROM simuladorarea WHERE id_area = '" + id_area + "'";  
        Connection cn = new Conexion(ipAddress).conectar();  
        PreparedStatement pstDelete = cn.prepareStatement(delete);  
  
        pstDelete.executeUpdate();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
}
```

### 8.3.8. Datos Personales

Nombre(s):	<input type="text"/>	Actualizar
Apellido(s):	<input type="text"/>	Actualizar
Nombre de usuario:	<input type="text"/>	Actualizar
Contraseña:	<input type="text"/>	Actualizar
Teléfono celular:	<input type="text"/>	Actualizar
Correo electrónico:	<input type="text"/>	Actualizar
Rol:	<input type="text"/>	
		Borrar cuenta

Esta clase cuenta con los siguientes objetos y variables:

```
public class DatosPersonalesPanel extends javax.swing.JPanel {  
  
    private String id_usuario = "";  
    private String user = "";  
    private String rolUser = "";  
  
    private String nombresViejos;  
    private String apellidosViejos;  
    private String usernameViejo;  
    private String telefonoViejo;  
    private String correoViejo;  
    private JFrame pantallaPrincipal;
```

- **Id\_usuario, user, rolUser.** Todas variables de tipo String.
- **nombresViejos, apellidosViejos, usernameViejo, contrasenaVieja, telefonoViejo y correoViejo.** Todas variables de tipo String. Sirven para comparar los datos ya existentes del usuario y ver si ha habido algún cambio con respecto a los ingresados en los campos de texto.
- **pantallaPrincipal,** objeto de la clase JFrame.

El método constructor recibe 4 argumentos: idUsuario (String), username (String), rol (String) y pantallaPrincipal (PantallaPrincipal).

Las variables nombresViejos, apellidosViejos, usernameViejo, contrasenaVieja, telefonoViejo y correoViejo reciben los valores de los campos de texto correspondientes.

Si el usuario tiene el rol de “MainAdmin”, se esconde el botón el botón para borrar su cuenta.



```
public DatosPersonalesPanel(String idUsuario, String username, String rol, JFrame pantallaPrincipal) {  
    initComponents();  
  
    this.id_usuario = idUsuario;  
    this.user = username;  
    this.rolUser = rol;  
    this.pantallaPrincipal = pantallaPrincipal;  
  
    if (rolUser.equals("MainAdmin")) {  
        botonBorrar.setVisible(false);  
    }  
  
    this.verDatos();  
  
    this.nombresViejos = textNombres.getText().trim();  
    this.apellidosViejos = textApellidos.getText().trim();  
    this.usernameViejo = textUsername.getText().trim();  
    this.telefonoViejo = textTelefono.getText().trim();  
    this.correoViejo = textCorreo.getText().trim();  
}
```

También posee el método **getIconImage** para seleccionar y mostrar el ícono de la ventana.

```
public Image getIconImage() {  
    Image retValue = Toolkit.getDefaultToolkit()  
        .getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));  
    return retValue;  
}
```

El método **verDatos** recoge todos los datos del usuario y los plasma en los campos de texto correspondientes.

```
private void verDatos() {  
    try{  
        String datos = "SELECT * FROM usuarios WHERE username = '" + user + "'";  
  
        Connection cn = new Conexion(ipAddress).conectar();  
  
        PreparedStatement pstDatos = cn.prepareStatement(datos);  
  
        ResultSet rsDatos = pstDatos.executeQuery();  
  
        while(rsDatos.next()) {  
            textNombres.setText(rsDatos.getString("nombres"));  
            textApellidos.setText(rsDatos.getString("apellidos"));  
            textUsername.setText(rsDatos.getString("username"));  
            textContrasena.setText(rsDatos.getString("contrasenia"));  
            textTelefono.setText(rsDatos.getString("telefono"));  
            textCorreo.setText(rsDatos.getString("correo"));  
            textRol.setText(rsDatos.getString("rol"));  
        }  
  
        cn.close();  
        pstDatos.close();  
        rsDatos.close();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
}
```

Existen una serie de métodos para cada botón de actualizar que se encargan de actualizar cada dato.

#### 1) btnActualizarNombresActionPerformed

```
private void btnActualizarNombresActionPerformed(java.awt.event.ActionEvent evt) {
    String nombres = textNombres.getText().trim();

    if (nombres.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Campo vacio.");
        return;
    }

    if (nombres.equals(nombresViejos)) {
        JOptionPane.showMessageDialog(this, "No se han cambiado datos.");
        return;
    }

    int yes_no = JOptionPane.showConfirmDialog(null,
        "¿Seguro que quiere actualizar sus nombres?", "Alerta", JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE);

    if (yes_no == 1) {
        return;
    }

    try {
        String checkUser = "SELECT * FROM usuarios WHERE id_usuario = " + id_usuario + "";

        Connection cn = new Conexion().conectar();
        PreparedStatement pst = cn.prepareStatement(checkUser);
        ResultSet rs = pst.executeQuery();

        if (rs.next()) {
            String updateQuery = "UPDATE usuarios SET nombres = ? WHERE id_usuario = " + id_usuario + "";
            PreparedStatement pstUpdateNombre = cn.prepareStatement(updateQuery);

            pstUpdateNombre.setString(1, nombres);

            pstUpdateNombre.executeUpdate();

            JOptionPane.showMessageDialog(this, "Nombres actualizados.");

            pstUpdateNombre.close();

            this.nombresViejos = textNombres.getText().trim();
        }

        cn.close();
        pst.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

## 2) btnActualizarApellidosActionPerformed

```
private void btnActualizarApellidosActionPerformed(java.awt.event.ActionEvent evt) {
    String apellidos = textApellidos.getText().trim();

    if (apellidos.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Campo vacio.");
        return;
    }

    if (apellidos.equals(apellidosViejos)) {
        JOptionPane.showMessageDialog(this, "No se han cambiado datos.");
        return;
    }

    int yes_no = JOptionPane.showConfirmDialog(null,
        "¿Seguro que quiere actualizar sus apellidos?", "Alerta", JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE);

    if (yes_no == 1) {
        return;
    }

    try {
        String checkUser = "SELECT * FROM usuarios WHERE id_usuario = '" + id_usuario + "'";

        Connection cn = new Conexion().conectar();
        PreparedStatement pst = cn.prepareStatement(checkUser);
        ResultSet rs = pst.executeQuery();

        if (rs.next()) {
            String updateQuery = "UPDATE usuarios SET apellidos = ? WHERE id_usuario = " + id_usuario + "";
            PreparedStatement pstUpdateNombre = cn.prepareStatement(updateQuery);

            pstUpdateNombre.setString(1, apellidos);

            pstUpdateNombre.executeUpdate();

            JOptionPane.showMessageDialog(this, "Datos actualizados.");

            pstUpdateNombre.close();

            this.apellidosViejos = textApellidos.getText().trim();
        }

        cn.close();
        pst.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

### 3) btnActualizarUsernameActionPerformed

```
private void btnActualizarUsernameActionPerformed(java.awt.event.ActionEvent evt) {
    String username = textUsername.getText().trim();

    if (username.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Campo vacío.");
        return;
    }

    if (username.equals(usernameViejo)) {
        JOptionPane.showMessageDialog(this, "No se han cambiado datos.");
        return;
    }

    int yes_no = JOptionPane.showConfirmDialog(null,
        "¿Seguro que quiere actualizar su nombre de usuario?", "Alerta", JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE);

    if (yes_no == 1) {
        return;
    }

    String password = confirmPassword();
    String hashedPassword = Passwords.cipher(password, usernameViejo);

    if (!hashedPassword.equals(Passwords.getPassword(id_usuario))) {
        JOptionPane.showMessageDialog(this, "Contraseña incorrecta.");
        return;
    }

    try {
        String checkUsername = "SELECT * FROM usuarios WHERE username = '" + username + "'";

        Connection cn = new Conexion().conectar();
        PreparedStatement pst = cn.prepareStatement(checkUsername);
        ResultSet rs = pst.executeQuery();

        if (!rs.next()) {
            String updateQuery = "UPDATE usuarios SET username = ?, contrasenia = ? WHERE id_usuario = " + id_usuario;
            PreparedStatement pstUpdateUsername = cn.prepareStatement(updateQuery);

            String newHashedPassword = Passwords.cipher(password, username);

            pstUpdateUsername.setString(1, username);
            pstUpdateUsername.setString(2, newHashedPassword);

            pstUpdateUsername.executeUpdate();

            JOptionPane.showMessageDialog(this, "Datos actualizados.");

            pstUpdateUsername.close();

            this.usernameViejo = textUsername.getText().trim();
        } else {
            JOptionPane.showMessageDialog(this, "Ya existe un usuario con este nombre de usuario.");
        }

        cn.close();
        pst.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

#### 4) btnActualizarPassword

```
private void btnActualizarPasswordActionPerformed(java.awt.event.ActionEvent evt) {  
    String password = passwordField.getText().trim();  
    String hashedNewPassword = Passwords.cipher(password, usernameViejo);  
  
    if (password.isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Campo vacío.");  
        this.passwordField.setText("");  
        return;  
    }  
  
    if (!Validaciones.validatePassword(password)) {  
        JOptionPane.showMessageDialog(this, "Contraseña no válida. Debe haber mínimo "  
            + "8 caracteres, una o más mayúsculas y minúsculas, y uno "  
            + "o más números");  
        this.passwordField.setText("");  
        return;  
    }  
  
    int yes_no = JOptionPane.showConfirmDialog(null,  
        "¿Seguro que quiere actualizar su contraseña?", "Alerta", JOptionPane.YES_NO_OPTION,  
        JOptionPane.WARNING_MESSAGE);  
  
    if (yes_no == 1) {  
        this.passwordField.setText("");  
        return;  
    }  
  
    if (hashedNewPassword.equals(Passwords.getPassword(id_usuario))) {  
        JOptionPane.showMessageDialog(this, "Misma contraseña.");  
        this.passwordField.setText("");  
        return;  
    }  
}
```

```
String confirmPassword = confirmPassword();
String hashedConfirmPassword = Passwords.cipher(confirmPassword, usernameViejo);

if (!hashedNewPassword.equals(hashedConfirmPassword)) {
    JOptionPane.showMessageDialog(this, "Las contraseñas no coinciden.");
    this.passwordField.setText("");
    return;
}

try {
    String checkUser = "SELECT * FROM usuarios WHERE id_usuario = '" + id_usuario + "'";

    Connection cn = new Conexion().conectar();
    PreparedStatement pst = cn.prepareStatement(checkUser);
    ResultSet rs = pst.executeQuery();

    if (rs.next()) {
        String updateQuery = "UPDATE usuarios SET contrasenia = ? WHERE id_usuario = " + id_usuario + " ";
        PreparedStatement pstUpdatePassword = cn.prepareStatement(updateQuery);

        pstUpdatePassword.setString(1, hashedNewPassword);

        pstUpdatePassword.executeUpdate();

        JOptionPane.showMessageDialog(this, "Datos actualizados.");

        pstUpdatePassword.close();

        this.passwordField.setText("");
    }

    cn.close();
    pst.close();
}
```

## 5) btnActualizarTelefonoActionPerformed

```
private void btnActualizarTelefonoActionPerformed(java.awt.event.ActionEvent evt) {
    String telefono = textTelefono.getText().trim();

    if (telefono.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Campo vacío.");
        return;
    }

    if (telefono.equals(telefonoViejo)) {
        JOptionPane.showMessageDialog(this, "No se han cambiado datos.");
        return;
    }

    if (!Validaciones.validatePhone(telefono)) {
        JOptionPane.showMessageDialog(this, "Telefono no válido.");
        return;
    }

    int yes_no = JOptionPane.showConfirmDialog(null,
        "¿Seguro que quiere actualizar su telefono?", "Alerta", JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE);

    if (yes_no == 1) {
        return;
    }

    try {
        String checkTelefono = "SELECT * FROM usuarios WHERE telefono = '" + telefono + "'";

        Connection cn = new Conexion().conectar();
        PreparedStatement pst = cn.prepareStatement(checkTelefono);
        ResultSet rs = pst.executeQuery();

        if (!rs.next()) {
            String updateQuery = "UPDATE usuarios SET telefono = ? WHERE id_usuario = " + id_usuario + ";";
            PreparedStatement pstUpdateTelefono = cn.prepareStatement(updateQuery);

            pstUpdateTelefono.setString(1, telefono);

            pstUpdateTelefono.executeUpdate();

            JOptionPane.showMessageDialog(this, "Teléfono actualizado.");

            pstUpdateTelefono.close();

            this.telefonoViejo = textTelefono.getText().trim();
        } else {
            JOptionPane.showMessageDialog(this, "Ya existe alguien con este teléfono.");
        }

        cn.close();
        pst.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



## 6) btnActualizarCorreoActionPerformed

```
private void btnActualizarCorreoActionPerformed(java.awt.event.ActionEvent evt) {
    String correo = textCorreo.getText().trim();

    if (correo.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Campo vacío.");
        return;
    }

    if (correo.equals(correoViejo)) {
        JOptionPane.showMessageDialog(this, "No se han cambiado datos.");
        return;
    }

    if (!Validaciones.validateEmail(correo)) {
        JOptionPane.showMessageDialog(this, "Correo no válido.");
        return;
    }

    int yes_no = JOptionPane.showConfirmDialog(null,
        "¿Seguro que quiere actualizar su correo?", "Alerta", JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE);

    if (yes_no == 1) {
        return;
    }

    try {
        String checkCorreo = "SELECT * FROM usuarios WHERE correo = '" + correo + "'";

        Connection cn = new Conexion().conectar();
        PreparedStatement pst = cn.prepareStatement(checkCorreo);
        ResultSet rs = pst.executeQuery();

        if (!rs.next()) {
            String updateQuery = "UPDATE usuarios SET correo = ? WHERE id_usuario = " + id_usuario + "";
            PreparedStatement pstUpdateCorreo = cn.prepareStatement(updateQuery);

            pstUpdateCorreo.setString(1, correo);

            pstUpdateCorreo.executeUpdate();

            JOptionPane.showMessageDialog(this, "Correo actualizado.");

            pstUpdateCorreo.close();

            this.correoViejo = textCorreo.getText().trim();
        } else {
            JOptionPane.showMessageDialog(this, "Ya existe alguien con este correo.");
        }

        cn.close();
        pst.close();
        rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

El método **botonBorrarActionPerformed** se llama al presionar el botón “Borrar cuenta” y elimina al usuario de la base de datos.

```
private void botonBorrarActionPerformed(java.awt.event.ActionEvent evt) {  
  
    int yes_no = JOptionPane.showConfirmDialog(null,  
        "¿Seguro que quiere eliminar su cuenta?", "Alerta", JOptionPane.YES_NO_OPTION,  
        JOptionPane.WARNING_MESSAGE);  
    if(yes_no == 0){  
        try{  
            String deleteUser = "DELETE FROM usuarios WHERE id_usuario = '" + id_usuario + "'";  
  
            Connection cn = new Conexion(ipAddress).conectar();  
            PreparedStatement pstDelete = cn.prepareStatement(deleteUser);  
  
            pstDelete.executeUpdate();  
            JOptionPane.showMessageDialog(null, "Cuenta eliminada exitosamente.");  
  
            cn.close();  
            pstDelete.close();  
  
            this.dispose();  
            this.pp.dispose();  
            new InicioSesion().setVisible(true);  
        }  
        catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

El último método de la clase, **confirmPassword**, se encarga de que el usuario confirme su contraseña en caso de que quiera actualizar su username o contraseña.

```
public String confirmPassword() {  
    JPasswordField myPasswordField = new JPasswordField(24);  
    Box box = Box.createHorizontalBox();  
    box.add(myPasswordField);  
  
    int x = JOptionPane.showConfirmDialog(null, box, "Confirme contraseña", JOptionPane.OK_CANCEL_OPTION);  
  
    if (x == JOptionPane.OK_OPTION) {  
        return myPasswordField.getText().trim();  
    }  
  
    return null;  
}
```

### 8.3.9. VerUsuariosPanel

Repositorio de Minería

Nombre	Username	Teléfono	Correo	Rol
Carlos Alberto Gonzalez Guerterito	Yagatomon	8331112234	correoverdadero@hotmail.com	MainAdmin
Alvin Yakitori	Adminn	(833) 111 1234	correoalgo@hotmail.com	Admin
Algo Probando	Moomers	8331998828	email@gmail.com	Usuario

Menu

Usuario seleccionado: Adminn

Rol: Admin

Actualizar rol

Eliminar usuario

Las variables y objetos de esta clase son los siguientes:

```
private String headerUsuarios[] = {"Nombre", "Username", "Teléfono", "Correo", "Rol"};
private DefaultTableModel tableModelUsuarios = new DefaultTableModel(headerUsuarios, 0);
private JTable tablaUsuarios;
private String rolAntiguo = "";
private String user = "";
```

- **headerUsuarios**, un arreglo de tipo String que define las columnas de la tabla que se mostrará.
- **tableModelUsuarios**, el modelo de la tabla que toma como base el arreglo headerUsuarios.
- **tablaUsuarios**, un objeto tipo JTable.
- **rolAntiguo**, una variable de tipo String que mantendrá control del rol del usuario antes de que este sea cambiado.

- **user**, una variable de tipo String.

También posee el método **getIconImage** para seleccionar y mostrar el ícono de la ventana.

```
public Image getIconImage() {
    Image retValue = Toolkit.getDefaultToolkit()
        .getImage(ClassLoader.getResource("imagenes/cascoIcon.png"));
    return retValue;
}
```

El método constructor recibe solo un argumento: username (String).

```
public VerUsuariosPanel(String username) {
    initComponents();

    this.user = username;

    comboBoxRol.removeAllItems();

    comboBoxRol.addItem("-");
    comboBoxRol.addItem("MainAdmin");
    comboBoxRol.addItem("Admin");
    comboBoxRol.addItem("Usuario");
    comboBoxRol.setEnabled(false);

    tablaUsuarios = verTablaUsuarios();
    tablePanel.setLayout(new BorderLayout());
    JScrollPane scrollPane = new JScrollPane(tablaUsuarios);
    scrollPane.setBorder(BorderFactory.createEmptyBorder());
    tablePanel.add(scrollPane, BorderLayout.CENTER);
    tablePanel.revalidate();
    tablePanel.repaint();
}
```

Dentro del método constructor, se le asigna un MouseListener a *tablaUsuarios*. Si alguna de las filas es seleccionada, se hace lo siguiente:

1. Se obtiene el nombre de usuario y se muestra en una caja de texto.
2. Se guarda el valor del rol actual en la variable *rolAntiguo*.
3. El ComboBox selecciona el valor de *rolAntiguo*.

4. Se remueve el valor “nulo” del ComboBox (“-“).
5. Si el valor de *rolAntiguo* es “MainAdmin” y la cantidad de MainAdmins es menor o igual a 1 o el nombre de usuario es el mismo de la persona manejando el sistema actualmente, se deshabilitan los botones y el combo box, impidiendo el modificar la base de datos de los usuarios.

```
tablaUsuarios.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        textUsername.setText(tablaUsuarios.getValueAt(tablaUsuarios.getSelectedRow(), 1).toString());
        rolAntiguo = tablaUsuarios.getValueAt(tablaUsuarios.getSelectedRow(), 4).toString();
        comboBoxRol.setSelectedItem(rolAntiguo);
        comboBoxRol.removeItem("-");
        if(rolAntiguo.equals("MainAdmin") && contarMainAdmins() <= 1 || user.equals(textUsername.getText().trim())){
            comboBoxRol.setEnabled(false);
            botonEliminarUsuario.setEnabled(false);
            botonActualizarRol.setEnabled(false);
        }
        else{
            comboBoxRol.setEnabled(true);
            botonEliminarUsuario.setEnabled(true);
            botonActualizarRol.setEnabled(true);
        }
    }
});
```

Dentro del método constructor, si el número de usuarios en el sistema es 1, se inhabilita el botón de Eliminar.

```
if(contarUsuarios() == 1){
    botonEliminarUsuario.setEnabled(false);
}

this.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        pp.setEnabled(true);
    }
});
```

El método **contarMainAdmins** cuenta la cantidad de usuarios cuyo rol es igual a “MainAdmin”.

```
private int contarMainAdmins(){  
    int contador = 0;  
  
    try{  
        String consulta = "SELECT * FROM usuarios WHERE rol = 'MainAdmin';  
  
        Connection cn = new Conexion(ipAddress).conectar();  
        PreparedStatement pst = cn.prepareStatement(consulta);  
        ResultSet rs = pst.executeQuery();  
  
        while(rs.next()){  
            contador++;  
        }  
  
        cn.close();  
        pst.close();  
        rs.close();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
  
    return contador;  
}
```

El método **contarUsuarios**, por el contrario, cuenta a *todos* los usuarios registrados en el sistema.

```
private int contarUsuarios() {  
    int contador = 0;  
  
    try{  
        String consulta = "SELECT * FROM usuarios";  
  
        Connection cn = new Conexion(ipAddress).conectar();  
        PreparedStatement pst = cn.prepareStatement(consulta);  
        ResultSet rs = pst.executeQuery();  
  
        while(rs.next()) {  
            contador++;  
        }  
  
        cn.close();  
        pst.close();  
        rs.close();  
    }  
    catch(Exception e){  
        e.printStackTrace();  
    }  
  
    return contador;  
}
```

El método **botonActualizarRolActionPerformed** se ejecuta al presionar el botón “Actualizar rol” y hace lo siguiente:

1. Se asegura que haya seleccionado a un usuario. Si no lo hace, se arroja una advertencia.
2. Si no se selecciona un rol distinto, marca una advertencia.
3. De lo contrario, ejecuta la actualización y cambia el rol del usuario seleccionado.

```
private void botonActualizarRolActionPerformed(java.awt.event.ActionEvent evt) {  
    String username = textUsername.getText().trim();  
    String rol = comboBoxRol.getSelectedItem().toString();  
  
    if(rolAntiguo.equals("")){  
        JOptionPane.showMessageDialog(null, "Seleccione un usuario.");  
    }  
    else if(rolAntiguo.equals(rol)){  
        JOptionPane.showMessageDialog(null, "No se seleccionó un rol distinto.");  
    }  
    else{  
        try{  
            String updateRol = "UPDATE usuarios SET rol = ? WHERE username = '" + username + "'";  
  
            Connection cn = new Conexion(ipAddress).conectar();  
            PreparedStatement pstUpdateRol = cn.prepareStatement(updateRol);  
  
            pstUpdateRol.setString(1, rol);  
            pstUpdateRol.executeUpdate();  
  
            rolAntiguo = rol;  
  
            cn.close();  
            pstUpdateRol.close();  
  
            tableModelUsuarios.setRowCount(0);  
            verTablaUsuarios();  
  
            JOptionPane.showMessageDialog(null, "Rol actualizado correctamente.");  
        }  
        catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

El método **botonEliminarUsuarioActionPerformed** se ejecuta al presionar el botón “Eliminar usuario” y hace lo siguiente:

1. Se asegura que haya seleccionado a un usuario. Si no lo hace, se arroja una advertencia.
2. Elimina al usuario seleccionado.



```
private void botonEliminarUsuarioActionPerformed(java.awt.event.ActionEvent evt) {

    if(rolAntiguo.equals("")){
        JOptionPane.showMessageDialog(null, "Seleccione un usuario.");
    }
    else{
        int yes_no = JOptionPane.showConfirmDialog(null,
            "¿Seguro que quiere eliminar este usuario?", "Alerta", JOptionPane.YES_NO_OPTION,
            JOptionPane.WARNING_MESSAGE);

        if(yes_no == 0){
            String username = textUsername.getText().trim();

            try{
                String deleteUser = "DELETE FROM usuarios WHERE username = '" + username + "'";

                Connection cn = new Conexion(ipAddress).conectar();
                PreparedStatement pstDelete = cn.prepareStatement(deleteUser);

                pstDelete.executeUpdate();
                JOptionPane.showMessageDialog(null, "Usuario eliminado exitosamente.");

                cn.close();
                pstDelete.close();

                tableModelUsuarios.setRowCount(0);
                verTablaUsuarios();

                rolAntiguo = "";
            }
            catch(Exception e){
                e.printStackTrace();
            }
        }
    }
}
```

El siguiente método en la clase es **verTablaUsuarios**. Este método se encarga de recolectar los nombres, apellidos, username, teléfono, correo y rol de cada usuario para plasmarlo en la tabla de la interfaz.

```
private JTable verTablaUsuarios(){
    JTable tabla = new JTable(){
        @Override
        public Component prepareRenderer(TableCellRenderer renderer, int row, int column){
            Component component = super.prepareRenderer(renderer, row, column);
            int rendererWidth = component.getPreferredSize().width;
            TableColumn tableColumn = getColumnModel().getColumn(column);
            tableColumn.setPreferredWidth(Math.max(rendererWidth + getInterCellSpacing().width,
                tableColumn.getPreferredWidth()));
            return component;
        }
    };
};
```

El método empieza creando un objeto llamado “tabla” de la clase JTable, llamando al método **prepareRenderer**, el cual ayudará a ajustar el ancho de cada columna al contenido de las casillas.

```
TablaImagen imgRenderer = new TablaImagen();  
imgRenderer.setHorizontalAlignment(JLabel.CENTER);  
tabla.setDefaultRenderer(Object.class, imgRenderer);  
tabla.setRowHeight(60);  
tabla.setDefaultEditor(Object.class, null);  
tabla.setFillViewportHeight(true);  
tabla.getTableHeader().setReorderingAllowed(false);  
tabla.setFont(new Font("Arial", Font.BOLD, 16));  
tabla.getTableHeader().setOpaque(false);  
tabla.getTableHeader().setBackground(new Color(253,193,1));  
tabla.getTableHeader().setFont(new Font("Arial", Font.BOLD, 18));
```

Posteriormente, se definen las características de la tabla. Cada línea significa lo siguiente:

1. Creación de un objeto de la clase **TablaImagen** llamado “imgRenderer”, que permitirá poner JLabels, y por tanto, imágenes en las columnas.
2. Definir la alineación de imgRenderer. En este caso se puso como “CENTER”, para que el contenido de cada celda esté centrado.
3. Definir el renderizador de la tabla, en este caso, imgRenderer.
4. Se define la altura de cada fila (50 px).
5. Se define el editor de la tabla como *null* para evitar que los usuarios modifiquen los valores de cada columna al dar doble clic.
6. Se establece que la columna llenará el alto de cada fila.
7. Se impide que el usuario pueda alterar el orden de las columnas con el mouse.
8. Se define la tipografía de la tabla.
9. Se define que los encabezados de la tabla no serán opacos, para permitir cambiar sus colores.
10. Se define el color de los encabezados.
11. Se define la tipografía de los encabezados.

Posteriormente, se conecta a la base de datos y se recolectan los datos de cada usuario y se agregan como una fila a **tableModelSimuladores**.

```
String consulta = "SELECT CONCAT(nombres, ' ', apellidos) AS nombre, username, telefono, correo, rol "
                  + "FROM usuarios";

try{
    Connection cn = new Conexion(ipAddress).conectar();
    PreparedStatement pstConsulta = cn.prepareStatement(consulta);
    ResultSet rsConsulta = pstConsulta.executeQuery();

    while(rsConsulta.next()){
        Object fila[] = new Object[5];

        fila[0] = rsConsulta.getString("nombre");
        fila[1] = rsConsulta.getString("username");
        fila[2] = rsConsulta.getString("telefono");
        fila[3] = rsConsulta.getString("correo");
        fila[4] = rsConsulta.getString("rol");

        tableModelUsuarios.addRow(fila);
    }
    tabla.setModel(tableModelUsuarios);
    cn.close();
    pstConsulta.close();
    rsConsulta.close();
}
catch(Exception e){
    e.printStackTrace();
}

return tabla;
}
```