PRÁCTICA II

BASE DE DATOS II

CREACIÓN DE UNA RED SOCIAL UNIVERSITARIA

CURSO 2022/2023

AUTORES

Mario Ventura Burgos

DNI - **43223476J**

Jose Enrique Sánchez Weiss

DNI - **43475994Z**

Sasha Cammarata San Segundo

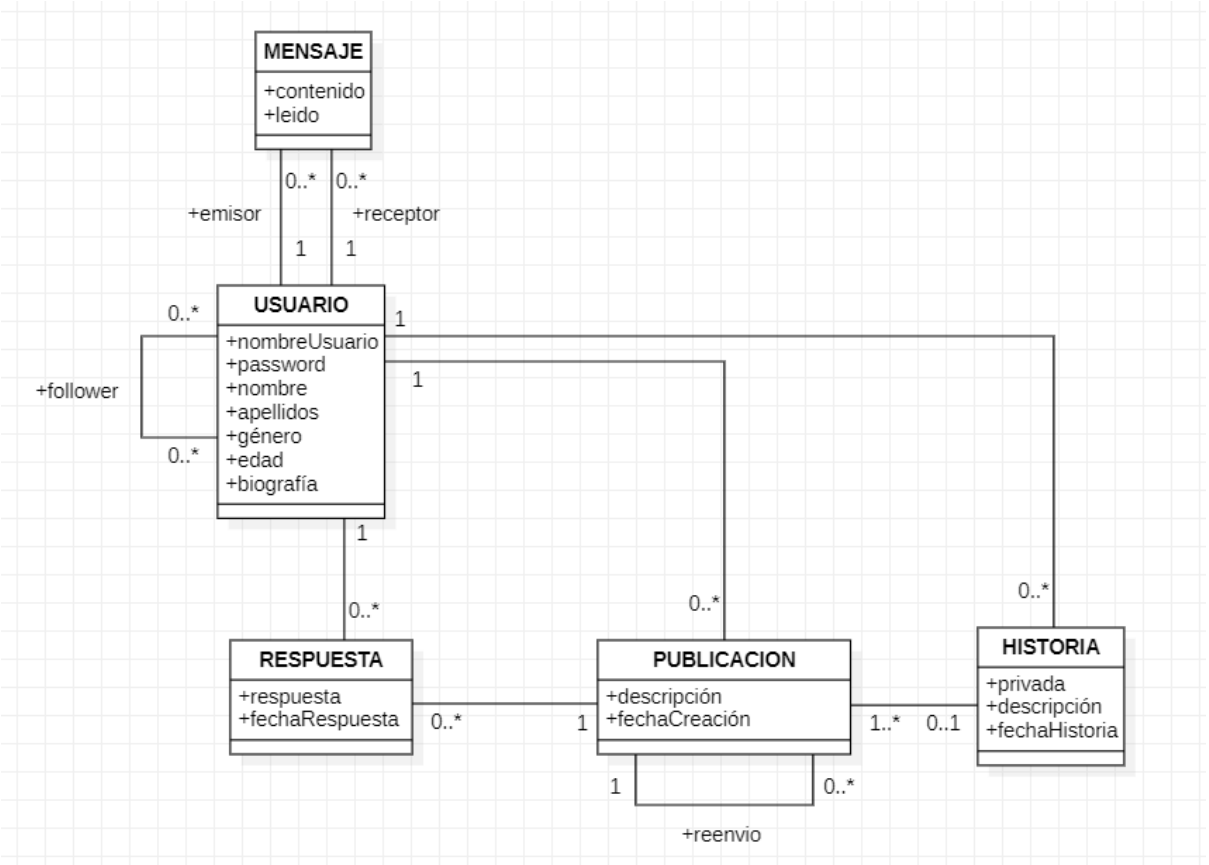
DNI - **43224428E**

Arya Mangunsatya I Gusti Ngurah Agung

NIE - **X7682807W**

ÍNDICE

1. MODELO CONCEPTUAL



1.1 - SUPOSICIONES Y EXPLICACIONES

MENSAJE:

Esta clase contiene el **texto** que se desea enviar o el texto recibido por un usuario y un booleano para comprobar si el mensaje ha sido leído o no. Hemos asumido que un usuario puede haber enviado o recibido 0 o más mensajes, y cada mensaje tiene asociados **un único usuario receptor** y **un único usuario emisor** (desde la tabla mensaje se podrá acceder a los datos tanto del usuario emisor como receptor).

USUARIO:

Esta clase contiene el **nombre** y la **contraseña** de un usuario concreto, así como otros datos personales como su **nombre**, **apellidos**, **género**, **edad** y una breve **biografía** . Hemos asumido que un usuario concreto tiene asociadas **cero o más publicaciones** (podría no haber publicado nada todavía), **cero o más respuestas** (tal vez no haya respondido a ninguna publicación) y **cero o más historias** (es

posible que un usuario no haya creado ninguna historia). Además, también hemos añadido una relación reflexiva con la tabla usuario que representa el hecho de que un usuario puede ser follower de 0 o más usuarios, y un usuario puede tener 0 o más followers.

RESPUESTA:

Esta clase representa la respuesta de un usuario a una determinada publicación, y contiene tanto el **texto** de la respuesta en sí como la **fecha de publicación** de dicha respuesta. Hemos asumido que una respuesta va asociada a **un único usuario** y a **una única publicación**, y desde la tabla Respuesta se puede acceder tanto a los datos de la publicación y del usuario.

PUBLICACIÓN:

Esta clase representa una publicación de un usuario en nuestra red social y contiene el texto de la publicación y su fecha de publicación. Hemos supuesto que cada publicación puede tener **cero o más respuestas,** puede estar incluída en **cero o más historias** y van asociadas a **un único usuario**. Además, también hemos incluido en esta clase una relación reflexiva, que representa el hecho de que **una única publicación** puede haber sido reenviada **0 o más veces**. Desde la tabla Publicación se puede acceder tanto a las **historias** como a los **usuarios**, así como a la **publicación** que se haya re-enviado en caso de que sea así.

HISTORIA:

Esta clase representa una publicación del tipo historia. Una historia puede ser **privada o pública** (booleano), tiene una **descripción** y una fecha de **publicación**.

Una historia concreta puede ser publicada solamente **por 1 usuario**, por tanto, la relación que une a ambas clases contendrá una multiplicidad de 0..* a 1.

Por otro lado, una historia es un conjunto de un **mínimo de 1 publicación**. Dado que **no existe un máximo** de publicaciones contenidas en una historia, la relación que une estas dos clases tendrá la multiplicidad 0..1 a 1..* (el 0..1 proviene de la suposición de que una publicación puede pertenecer o no a una historia).

3.

2. MODELO RELACIONAL

Se resaltará las Primary Key en **negrita**.

Se resaltará las Foreign Key en *cursiva.*

A continuación se explicará el proceso o pasos seguidos para crear el modelo relacional sobre el cuál se basará la base de datos final:

2.1 - PASOS DEL RELACIONAL

➔ 2.1.1 - PASO 1 & 2

Identificamos todas las clases, que pasarán a ser una tabla del modelo relacional, de nuestro modelo conceptual y definimos su clave primaria **PK** para cada una de ellas. En el caso de aquellas clases que cuenten con un atributo único serán definidas como PK pero en caso contrario, se creará un atributo **id** de aquella clase que será la clave primaria:

MENSAJE(**idMensaje,** contenido, leído)

USUARIO(**nombreUsuario,** password, nombre, apellidos, género, edad, biografía)

RESPUESTA(**idRespuesta,** respuesta, fechaRespuesta)

PUBLICACION(**idPublicación,** descripción, fechaCreación)

HISTORIA(**idHistoria,** privada, descripción, fechaHistoria)

➔ 2.1.2 - PASO 3 & 4

Identificamos todas las relaciones de nuestro modelo conceptual, que pasarán a ser una tabla del modelo relacional y de cada nueva tabla seleccionamos una clave primaria. En el caso de que la cardinalidad de una relación entre dos clases sea de muchos a muchos, se cogerán como PK las claves primarias de ambas clases.

R_EMISOR(**idMensaje,** nombreEmisor)

R_RECEPTOR(**idMensaje,** nombreReceptor)

R_FOLLOWER**(nombreUsuario1, nombreUsuario2**)

R_USUARIO_RESPUESTA(**idRespuesta,** nombreUsuario)

R_RESPUESTA_PUBLICACION(**idRespuesta,** idPublicación)

R_REENVIO(**idPublicación1,** idPublicación2)

R_PUBLICACION_HISTORIA(**idPublicación,** idHistoria)

R_USUARIO_PUBLICACION(**idPublicacion,** nombreUsuario)

R_USUARIO_HISTORIA(**idHistoria,** nombreUsuario)

➔ 2.1.3 - PASO 5

Se estudia la posibilidad de qué tablas se pueden fusionar de las tablas resultantes. De las tablas con misma clave se mira si son candidatas según su significado y conveniencia.

MENSAJE(**idMensaje,** contenido, leído, *nombreEmisor, nombreReceptor*)

USUARIO(**nombreUsuario,** password, nombre, apellidos, género, edad, biografía)

RESPUESTA(**idRespuesta,** respuesta, fechaRespuesta, *nombreUsuario, idPublicación)*

PUBLICACION(**idPublicacion,** descripción, fechaCreación, *nombreUsuario, idHistoria, idPublicación2)*

HISTORIA(**idHistoria,** privada, descripción, fechaHistoria, *nombreUsuario)*

R_FOLLOWER**(nombreUsuario1, nombreUsuario2**)

2.2 - SUPOSICIONES Y EXPLICACIONES

- **MENSAJE** se fusiona con **R_EMISOR** y **R_RECEPTOR,** obteniendo así dos foreign keys a la clase **USUARIO.**
- **USUARIO** no se fusiona.
- **RESPUESTA** se fusiona con **R_USUARIO_RESPUESTA** y con **R_RESPUESTA_PUBLICACION,** obteniendo de esta manera una foreign key a **USUARIO** y una a **PUBLICACION,** respectivamente.
- **PUBLICACION** se fusiona con **R_USUARIO_PUBLICACION, R_PUBLICACION_HISTORIA** y con **R_REENVIO**, obteniendo de esta forma una foreign key a **USUARIO,** una a **HISTORIA** y una a la misma clase PUBLICACION para poder representar que la publicación realizada pueda ser un posible reenvío en este caso.
- **HISTORIA** se fusiona con **R_HISTORIA_USUARIO** para obtener de esta manera una foreign key a la clase **USUARIO**.
- **R_FOLLOWER,** al no fusionarse con ninguna tabla, se mantiene.

3. CÓDIGO DE CREACIÓN BBDD

```
CREATE DATABASE bd203;


CREATE TABLE usuario(
    nombreUsuario char(20) NOT NULL,
    password char(20) NOT NULL,
    nombre char(15) NOT NULL,
    apellidos char(25),
    genero ENUM('Hombre','Mujer'),
    edad INTEGER NOT NULL,
    biografia char(150),
    PRIMARY KEY(nombreUsuario)
    );

CREATE TABLE mensaje(
    idMensaje INTEGER AUTO_INCREMENT NOT NULL,
    contenido char(180) NOT NULL,
    leido BOOLEAN NOT NULL,
    nombreEmisor char(20) NOT NULL,
    nombreReceptor char(20) NOT NULL,
    PRIMARY KEY (idMensaje),
    FOREIGN KEY (nombreEmisor) REFERENCES usuario(nombreUsuario),
    FOREIGN KEY (nombreReceptor) REFERENCES usuario(nombreUsuario)
    );

CREATE TABLE historia(
    idHistoria INTEGER AUTO_INCREMENT NOT NULL,
    privada BOOLEAN NOT NULL,
    descripcion char(255),
    fechaHistoria DATETIME NOT NULL,
    nombreUsuario char(20) NOT NULL,
    PRIMARY KEY(idHistoria),
    FOREIGN KEY (nombreUsuario) REFERENCES usuario(nombreUsuario)
    );

CREATE TABLE publicacion(
    idPublicacion INTEGER AUTO_INCREMENT NOT NULL,
    descripcion char(255),
    fechaCreacion DATETIME NOT NULL,
    nombreUsuario char(20) NOT NULL,
    idHistoria INTEGER,
    idPublicacion2 INTEGER,
    PRIMARY KEY (idPublicacion),
    FOREIGN KEY (nombreUsuario) REFERENCES usuario(nombreUsuario),
    FOREIGN KEY (idHistoria) REFERENCES historia(idHistoria),
    FOREIGN KEY (idPublicacion2) REFERENCES publicacion(idPublicacion)
    );

CREATE TABLE respuesta(
    idRespuesta INTEGER AUTO_INCREMENT NOT NULL,
    respuesta char(255) NOT NULL,
    fechaRespuesta DATETIME NOT NULL,
    nombreUsuario char(20) NOT NULL,
    idPublicacion INTEGER NOT NULL,
    PRIMARY KEY(idRespuesta),
    FOREIGN KEY (idPublicacion) REFERENCES publicacion(idPublicacion),
    FOREIGN KEY (nombreUsuario) REFERENCES usuario(nombreUsuario)
```

```
    );

CREATE TABLE r_follower(
    nombreUsuario1 char(20) NOT NULL,
    nombreUsuario2 char(20) NOT NULL,
    PRIMARY KEY(nombreUsuario1, nombreUsuario2),
    FOREIGN KEY (nombreUsuario1) REFERENCES usuario(nombreUsuario),
    FOREIGN KEY (nombreUsuario2) REFERENCES usuario(nombreUsuario)
    );

TABLAS PARA EL BACKUP:

TODAS LAS TABLAS SON EXACTAMENTE IGUALES, PERO TIENEN _BACKUP AL FINAL Y NO
HAY REFERENCIAS A FOREIGN KEYS PARA EVITAR PROBLEMAS. SON SIMPLEMENTE DATOS
RECUPERABLES.
```

Teniendo en cuenta todos los datos anteriores, se tomarán las siguientes suposiciones en cuanto a la implementación con php:

- En la tabla r_follower el atributo nombreUsuario1 sería el usuario que sigue y nombreUsuario2 sería el usuario al que siguen. Es decir, si ‘marioventura’ está en nombreUsuario1 y ‘joseenrique’ está en nombreUsuario2, el usuario ‘marioventura’ sigue a ‘joseenrique’.
- En la tabla publicación, si una publicación tiene el atributo idHistoria nulo, no estará dentro de una historia. En caso contrario, formará parte de una historia, y lo mismo con el atributo idPublicacion2. Si es nulo, no se tratará de una publicación reenviada y en caso contrario será una publicación reenviada.

4. MANUAL DE USUARIO

En este apartado cada integrante del grupo explicará su implementación de cada una de las partes de la red social y mostrará cómo el usuario puede interactuar con la propia red.

4.1 - INICIO de SESIÓN & HISTORIAS

Arya Mangunsatya I Gusti Ngurah Agung

➜ Para el **inicio de sesión**: El usuario dispondrá de la típica pantalla de *login* donde tendrá que rellenar los campos de un formulario con su nombre de usuario y contraseña en caso de que ya posean una cuenta registrada. En caso contrario, tendrán que crear una cuenta. Dicho formulario se realiza a través de un *form* en HTML al cual pasará la información introducida a un archivo PHP *iniciosesion.php* a través del método **post**.

En *iniciosesion.php*, recuperamos los datos y las asignamos a variables a través de **$_POST** y con dichos datos procedemos a generar la consulta SQL. Comprobamos si existe ese usuario en la base de datos y en caso positivo, podremos empezar una **sesión** PHP con **session_start()** donde guardaremos en las variables **$_SESSION** tanto el nombre de usuario como la contraseña y acto seguido, el usuario será redirigido a la página principal. En caso contrario, se mostrará un mensaje avisando que el usuario no existe y volverá a la pantalla de inicio de sesión.

En *principal.php*, mientras la sesión esté activa, cogemos el nombre de usuario guardada en la variable **$_SESSION[“nombreUsuario”]** y, mientras el usuario esté identificado todo este rato, podrá hacer uso de las diferentes funcionalidades de la red social como ver su perfil, crear una publicación, ver sus seguidores, etc. El usuario también puede cerrar sesión destruyendo la sesión PHP con **session_destroy()**.

➜ Para la **creación de historias**: El usuario se ha de dirigir al apartado de **Mis historias** en la barra de navegación de la página principal y le redirigirá a la página asociada al archivo PHP *historias.php* donde podrá crear una historia y consultar las historias que vaya creando.

Si el usuario desea crear una historia, debe de pulsar el botón “Crear” que lo redirigirá a un formulario *form* cuya acción está asociada a *insertHistoria.php* en el que tendrá que elegir la privacidad de la historia mediante un *drop-down list* de dos opciones (**Pública** con *value* = 0 y **Privada** con *value* = 1), además de redactar una descripción. En *insertHistoria.php*, cogemos los datos introducidos con **$_POST** y los usamos para generar un *Insert* en la tabla “historia”.

Como una historia es un conjunto de publicaciones, el usuario cuando cree una publicación en la página principal, tiene una *drop-down list* cuyas opciones son las historias que haya creado. Las historias que salgan en la feed principal serán solamente las del usuario y la de la gente que sigue.

Cada historia tiene un botón de “Ver Publicaciones” para visualizar las publicaciones de dicha historia y cuando se pulsa, se envía el atributo “idHistoria” a *verHistoria.php* y se captura este atributo con **$_GET**, la asignamos a una variable y generamos la consulta en la tabla “publicacion” para coger las filas cuya columna “idHistoria” coincida con el atributo capturado. Comprobamos si la historia tiene publicaciones o no contabilizando las filas en base a la consulta y en caso de que no hayan, saltará un mensaje avisando de que no hay publicaciones todavía y redirigirá al usuario a la página principal.

4.2 - CREACIÓN de CUENTA & PERFILES

Sasha Cammarata San Segundo

➜ Para la **creación de cuenta:** En la pantalla de *login* hay un enlace bajo el formulario de inicio de sesión que pide al usuario crear una cuenta si no tiene ninguna. Si se pulsa este enlace, lleva directamente al archivo html *createAccountForm.html,* que es otro form básico con un botón para volver atrás al inicio de sesión y otro para hacer submit del form al rellenarse con la información de usuario que se pide; si éste se pulsa una vez esté todo rellenado, enviará todos los datos a un archivo PHP *insertUser.php* a través del método **post**.

En *insertUser.php* se recogen todos los datos asignando cada uno a una variable php mediante el método **$_POST,** y con ello se genera la sentencia SQL para

insertar en la tabla **USUARIO** un usuario con todos los datos introducidos, pero también se comprueba con un SELECT básico si el nombre de usuario introducido ya existe para no tener dos claves primarias únicas, que generaría error, y por tanto en este caso se informaría al usuario de que el nombre ya existe. Cuando acaba, si el nombre de usuario no existe aún, vuelve a la pantalla de *login.*

➜ Para los **perfiles:** Una vez en *principal.php,* el usuario puede consultar en los enlaces de una barra de navegación, entre otras cosas, una lista de los seguidores que tiene, una lista de los usuarios a los que sigue y una lista de todos los usuarios registrados para poder seguir a quien quiera. Cada uno lleva a *verMisFollowers.php, verFollowed.php* y *verUsuarios.php*, en los que simplemente se hace una consulta de SQL y se muestran los nombres de usuario respectivos como botones con un enlace que si se clican llevan al perfil del usuario, es decir, llevan al archivo *verPerfil.php*.

En *verPerfil.php* se obtiene por parámetro con **$_GET** el nombre de usuario pasado por parámetro desde el archivo anterior con '?', que es el usuario del que se quiere ver el perfil, y se muestra toda su información básica. También se hace **session_start()** para obtener el nombre de usuario que ha iniciado sesión, y se comparan ambos para saber si el usuario que ha iniciado sesión es el que ha accedido a su perfil (desde el enlace de "ver mi perfil" en la barra de navegación de la página principal o desde una publicación o historia del propio usuario que aparezca en la actividad de la página principal). Si se produce esto, se muestra un botón para modificar el perfil que lleva al archivo *modificarPerfil.php*. En caso contrario, saldrá un botón para seguir o para dejar de seguir al usuario. Si se pulsa siendo "Seguir", llevará al archivo *follow.php,* que recoge los datos pasados por parámetro con '?' e inserta con SQL en la tabla R_FOLLOWER una nueva entrada de manera que el usuario de la sesión siga al usuario del perfil, y se volverá el botón verde como "Siguiendo" al recargarse. Si se pulsa estando en "Siguiendo", llevará al archivo *unfollow.php,* que recoge los datos de la misma forma que con *follow.php* pero en este caso hará con SQL un DELETE para hacer en términos prácticos que el usuario de la sesión deje de seguir al usuario del perfil. Además de todo esto, también se muestra una feed de publicaciones y de historias pero sólo de este usuario, igual que en la página principal. Si el usuario que mira el perfil es el que ha

iniciado sesión, se muestran absolutamente todas las publicaciones e historias y pueden borrarse también si el usuario lo decide. En caso contrario, se mostrarán todas las publicaciones igualmente y se mirará si el usuario que inicia sesión sigue al usuario del perfil. En caso afirmativo, mostrará tanto las historias públicas como las privadas, sino, sólo mostrará las públicas, y no tendrá la opción de eliminar nada porque no es su perfil.

Finalmente, en *modificarPerfil.php,* se recogen los datos originales del usuario pasados por parámetro con '?', y se visualiza un form donde aparecen todos estos datos, y hay un botón para volver atrás al perfil y otro para actualizar los datos si ya se han rellenado todos los campos. Es un formulario muy simple parecido al de crear una cuenta, sólo que no se puede modificar el nombre de usuario para evitar problemas de clave primaria con otras clases. Una vez se pulse el botón de actualizar, llevará al archivo *userModif.php,* que con los datos que recibe del form y coge con **$_POST,** hará un UPDATE con SQL en la base de datos en la tabla USUARIO. Una vez finalizado, vuelve de nuevo al perfil.

4.3 - RESPUESTAS, REENVÍOS Y PUBLICACIONES

Mario Ventura Burgos.

➜ Para las **Respuestas**: El usuario dispondrá de un cuadro de texto en la zona inferior de cada publicación, donde si lo desea, podrá introducir un texto de respuesta a esta publicación. El texto podrá ser de un máximo de 255 caracteres.

Si se pulsa el botón de responder a una publicación, pero no se ha escrito ningún texto de respuesta en el cuadro de texto pertinente, saldrá un aviso para el usuario que le informará de que no es posible publicar una respuesta vacía (y evidentemente, esta respuesta no se insertará en la base de datos).

La inserción de respuestas sigue la misma lógica que el resto de inserciones. Cuando un usuario escriba un texto de respuesta y pulse el botón responder, esta acción está asociada a *insertRespuesta.php*, donde se hará un ***session_start()***, se cogerán los datos introducidos con **$_POST** y se usarán para generar un *String* que se utilizará para hacer un *Insert* en la tabla "Respuesta" mediante la instrucción ***mysqli_query()***.

Cuando se vea una publicación en la página principal, cada una tendrá también un botón “Ver Respuestas”, donde al pulsarlo, se mostrará una lista de todas las respuestas. Esto sucede gracias a *verRespuestas.php*, donde se obtiene el valor del id de la publicación cuyas respuestas se quieren ver mediante **$_GET**. Se genera un String y se hace la consulta. Acto seguido, comprobamos si hay respuestas a esta publicación contando la cantidad de resultados que nos ha devuelto la consulta. Si es 0, no hay respuestas y se informa al usuario, y en caso contrario, se listan las respuestas que se han hecho a esa publicación, ordenadas por fecha de forma ascendente.

➜ Para los **Reenvíos**: Un reenvío, es una interacción más sobre una publicación. De hecho, es una interacción de una publicación sobre sí misma, que da valor al atributo *idPublicacion2* de la tabla publicación.

En cada publicación, existe un botón de reenviar (sea una publicación tuya o de otro usuario, estas siempre se podrán reenviar) en la esquina inferior izquierda, que al ser pulsado provocará las acciones asociadas a *insertReenvio.php*.

En *insertReenvio.php* se hace una inserción siguiendo el planteamiento del resto de las inserciones explicadas anteriormente. Se hace un session_start(), se obtiene el valor de los atributos *idPublicacion* y descripción mediante un **$_GET**, y se usan esto valores para generar un String que se usará para hacer un insert en la tabla “Publicación” con la sentencia ***mysqli_query()***.

Al hacer un insert en la tabla “Publicación”, realmente lo que se está haciendo es crear una nueva publicación, con la diferencia de que en este caso, la nueva publicación tendrá el atributo *idPublicacion2* con un valor diferente de **NULL** (en concreto, el *idPublicacion* de la publicación que se ha reenviado será el *idPublicacion2* de la nueva publicación de reenvío).

Todo esto será mostrado al usuario, de forma que una publicación reenviada aparecerá con el botón de reenvío en verde para aquel usuario que ya la ha reenviado.

➜ Para las **Respuestas**: Al iniciar sesión en Unigram, se podrá ver un cuadro de texto (justo debajo de la barra de menú donde se listan los apartados de la red social). Encima de este cuadro aparece el título “Crear Publicación”, y debajo, se puede ver un botón “Publicar”. Si hay algo escrito en el cuadro de texto, cuando el usuario pulse el botón, se hará la publicación. Esto sucede gracias al archivo *insertPublicacion.php*, que es la acción asociada al botón.

Al igual que en las otras inserciones, lo que se hará será obtener (**$_POST**) el valor de los atributos necesarios para generar el string que se usará para hacer el insert en la base de datos mediante la instrucción **mysqli_query()**.

Cabe mencionar que, dado que una publicación puede pertenecer (o no) a una historia (esta elección se hace mediante un desplegable que, en caso de quererlo, te permite escoger la historia a la cual pertenecerá la publicación; con el requisito de que esta historia deberá haber sido creada con anterioridad). Esta es una información relevante en cuanto a la generación del string que se usará para la inserción, ya que le dará valor al atributo *idHistoria*. Se obtendrá el valor de este atributo, y si tras esto el valor es **NULL**, significará que la publicación no pertenece a ninguna historia. De lo contrario, el valor de la *id* será el de la historia escogida.

4.4 - ENVÍO de MENSAJES

José Enrique Sánchez Weiss.

El usuario dispondrá de una lista con todos los usuarios de la red social (además de un botón con el que volver a la pestaña principal de la red social) a excepción del usuario que ha iniciado sesión en la zona lateral izquierda de la página . Podrá seleccionar cualquier usuario de la lista y, una vez seleccionado uno de los usuarios, se desplegarán a la derecha dos paneles: uno en la parte superior con todos los mensajes que se han enviado y recibido de dicho usuario, y otro en la parte inferior en el que se podrá introducir texto y enviarlo como un mensaje.

Para mostrar todos los posibles usuarios a los que enviar mensajes, se hace uso de *Mensajes.php.* Aquí se recorrerán todos los usuarios excepto el que tenga iniciada

la sesión ordenados por nombre, y en caso de que el usuario recorrido haya enviado un mensaje y el usuario que ha iniciado sesión no lo haya leído (si la columna leído de la tabla mensajes tiene el valor falso).

Por otro lado, para abrir la pestaña de mensajes con un usuario concreto, se hace uso de *verMensajes.php,* Incluyéndose también la clase anterior (Mensajes.php) para poder cambiar de usuario en caso de que se desee. En este caso, se obtendrán los mensajes cuyo emisor sea el usuario con la sesion iniciada y el receptor sea el usuario seleccionado de la lista, ordenados por ID. En caso de que el mensaje lo haya enviado el usuario principal, se mostrará a la izquierda, y si lo ha enviado el otro usuario a la derecha. También cabe recalcar que cada mensaje enviado por el usuario principal contará con un indicador que indica si se ha leído dicho mensaje o no (✓ = no leído y ✓✓ = leído), y que en esta clase también se crea el campo de texto para introducir el mensaje que se desea enviar.

El texto que se introduzca en el campo de texto de envío de mensajes se tratará en *enviarMensajes.php,* donde se realiza un insert en las tabla mensaje con los datos dados por parámetro (el texto del mensaje, el del nombre del emisor y el del receptor). Una vez se añade el mensaje a la tabla, se actualiza la página para mostrar la lista de mensajes con el nuevo mensaje añadido.

5. IMPLEMENTACIONES EXTRA

Durante la creación del código PHP, HTML y CSS de la página web, hemos tenido la necesidad de implementar un total de 3 triggers, 1 procedure y 1 event para poder hacer todas las funcionalidades extra correctamente.

5.1. - TRIGGER PARA BORRAR PUBLICACIONES

Para poder eliminar publicaciones, se debe tener en cuenta que antes se deben borrar todas las respuestas que pueda llegar a tener, debido a que si se borra una publicación directamente y ésta contiene respuestas, saltará un error ya que la primary key que identifica a esa publicación dejaría de existir y la foreign key en la tabla de respuestas que referencia la publicación a la que pertenece sería inválida. Por tanto, mediante el uso de un cursor, tenemos el siguiente trigger que insertamos en la base de datos y se ejecuta antes de un DELETE en la tabla de publicaciones:

```
DELIMITER //
CREATE trigger borrarPublicacion
BEFORE DELETE on publicacion FOR EACH ROW
BEGIN
DECLARE acabar INT DEFAULT FALSE;
DECLARE resp_aux int;
DECLARE varcursor CURSOR for SELECT idRespuesta from respuesta where
idPublicacion = OLD.idPublicacion;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET acabar=TRUE;
OPEN varcursor;
etq: LOOP
    FETCH varcursor into resp_aux;
    if acabar then
        leave etq;
     end IF;
    DELETE FROM respuesta WHERE idRespuesta=resp_aux;
end loop;
CLOSE varcursor;
END;
```

5.2. - TRIGGER PARA BORRAR HISTORIAS

Siguiendo la misma lógica que con las respuestas, si se quiere borrar historias; puede haber historias con publicaciones, y por ello se debe insertar un trigger en la base de datos que se ejecute antes de hacer un DELETE en la tabla de historias eliminando así todas las publicaciones asociadas a la historia, y así, en caso de que

las publicaciones de las historias tengan respuestas, cada DELETE dentro de este trigger llamará al trigger anterior.

```
DELIMITER //
CREATE trigger borrarHistoria
BEFORE DELETE on historia FOR EACH ROW
BEGIN
DECLARE acabar INT DEFAULT FALSE;
DECLARE pub_aux int;
DECLARE varcursor CURSOR for SELECT idPublicacion from publicacion where
idHistoria = OLD.idHistoria;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET acabar=TRUE;
OPEN varcursor;
etq: LOOP
    FETCH varcursor into pub_aux;
    if acabar then
        leave etq;
     end IF;
    DELETE FROM publicacion WHERE idPublicacion=pub_aux;
end loop;
CLOSE varcursor;
END;
```

5.3. - TRIGGER PARA ENVIAR MENSAJES

Para poder implementar la función que envía un mensaje a todos los seguidores del usuario una vez éste publica una historia con un enlace a ella, implementamos un cursor que mediante un SELECT vaya eligiendo todos los seguidores del usuario en cuestión obtenido tras realizar un INSERT en la tabla de historias y que haga un INSERT en mensaje con cada seguidor como usuario receptor que contiene un enlace al archivo php que en nuestro caso nos permite visualizar las publicaciones de una historia.

```
DELIMITER //
CREATE TRIGGER notificahistoria
AFTER INSERT on historia FOR EACH ROW
BEGIN
DECLARE acabar INT DEFAULT FALSE;
DECLARE a char(20);
DECLARE varcursor CURSOR FOR SELECT nombreUsuario1 FROM r_follower WHERE
nombreUsuario2 = NEW.nombreUsuario;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET acabar=TRUE;
OPEN varcursor;
etq: LOOP
    FETCH varcursor INTO a;
    IF acabar THEN
    LEAVE etq;
        END IF;
        INSERT INTO mensaje VALUES(null,concat('Mira mi nueva historia:
http://localhost/bd203/BD2X7682807/verHistoria.php?id=',NEW.idHistoria),0,NE
W.nombreUsuario,a);
END LOOP;
```

```
CLOSE varcursor;
END;
```

5.4. - PROCEDURE Y EVENT PARA EL BACKUP

Para poder implementar la función que hace una copia de seguridad diaria de la base de datos, necesitaremos implementar un PROCEDURE, que haciendo uso de todas las tablas en la base de datos creada como bd203_backup, lo que hace es borrar todos los datos de estas tablas con TRUNCATE por el caso de haber datos anteriores, y hace un INSERT en ellas con los datos de las tablas originales, y también hacemos un EVENT que lo que hará será invocar al PROCEDURE cada 24 horas desde la hora en la que se crea por primera vez (en este caso fue a la 9:00 AM).

```
DELIMITER //
CREATE PROCEDURE backUp()
BEGIN
        TRUNCATE TABLE respuesta_backup;
        TRUNCATE TABLE mensaje_backup;
        TRUNCATE TABLE publicacion_backup;
        TRUNCATE TABLE historia_backup;
        TRUNCATE TABLE r_follower_backup;
        TRUNCATE TABLE usuario_backup;
        INSERT into usuario_backup SELECT * FROM usuario;
        INSERT into r_follower_backup SELECT * FROM r_follower;
        INSERT into historia_backup SELECT * FROM historia;
        INSERT into publicacion_backup SELECT * FROM publicacion;
        INSERT into mensaje_backup SELECT * FROM mensaje;
        INSERT into respuesta_backup SELECT * FROM respuesta;
    END;
DELIMITER //
CREATE EVENT crear_backup
ON SCHEDULE every 24 hour STARTS CURRENT_TIMESTAMP
DO EXECUTE backUp;
```

5.5. - RECARGA DE LA PÁGINA CON META

En el caso de la funcionalidad para recargar la consulta del perfil cada 30 segundos, simplemente se hace una sentencia con HTML que invoca un refresco de la página cada 30 segundos con el valor <meta> introducido dentro del <head>, tal que así:

```
<head>
        <title>Pagina principal</title>
        <link rel="stylesheet" type="text/css" href="principalstyle.css">
        <!-- SE RECARGA LA PÁGINA CADA 30 SEGUNDOS -->
        <meta http-equiv="refresh" content="30;url=principal.php"/>
</head>
```

La recarga de 30 segundos se ha implementado tanto en la página principal de la red social como en la consulta del perfil de los usuarios.