Making a login form - Written by Anthony Adams

When we think of creating a login system, we have to take into account security measures and why they are important.

1 - For a login to work, we have to store those passwords somewhere. IE: in the database.

2 - Bad actors may want to get access to the database so they can get said passwords.

3 - We could store passwords as plain text but if we did this, a hacker has 1 point of failure to breach, access to the DB. Also any system admin can plainly see the passwords. **Do you trust all system admins?**

4 - We could deploy some counter measures to obscure passwords and turn them into gibberish in case the DB was accessed. One way of doing this could be using an encryption. However, if we use an encryption we would need to use an exception key. This leaves us

with 2 points of failure. Still better than 1 point of failure but anyone with inside access for example could easily get the key. Or if a hacker gets access to part of the system, they could potentially find the key.

5 - We could use a hash. A hash does not work like an encryption. Rather, hashing uses an algorithm. To make a long story short, once the user enters the desired password, the system can use something like Argon2 or bcrypt to create a hash of the original password. Once it is hashed it cannot be un-hashed as it is a 1 way process. What the system would then do is use the hash to check against an entered password when logging in. If the 2 hashes match, the password has a high likely chance of being the same (so high that it is ultimately unlikely that they would not match) and the user is granted access. Another feature of Argon2 and bcrypt is that they are designed to be slow. This means that it protects from brute force attacks. With modern GPU's a hacker can attempt to try billions of passwords and see if they match. Argon2 and bcrypt slow this down

so that it makes the process too slow for the hacker to attack.

Unfortunately human nature dictates that most people use bad passwords. Things like password1234, admin, asdf1234 and so on are often used. Because of this fact there are lists out there of likely passwords. Lists like this can still be used to guess user passwords. So If an attacker has a list of the logins they can use a list like this to guess against possible passwords and gain access to *some* of the user accounts in any system.

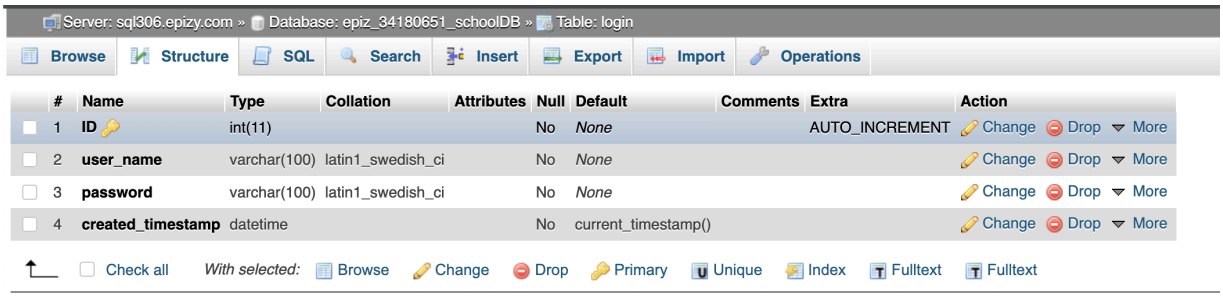
6 - We could mitigate further attacks with salting. Salting is the practice of appending a salt key before a hash. This throws an extra point of failure in front of the attacker. However, Argon2 hashing does have this feature built in so when using Argon2 or bscrypt you don't need to manually add a static salt, though you technically still could if you wanted to.

Salting could include static salting where you manually make a salt variable and add some

string of letters and numbers to it. Or using a random salting method of some kind. Both bscrypt and Argon2 have this built in.

The DB

We need to make a DB for the login system. We can then send user data to it when they sign up.



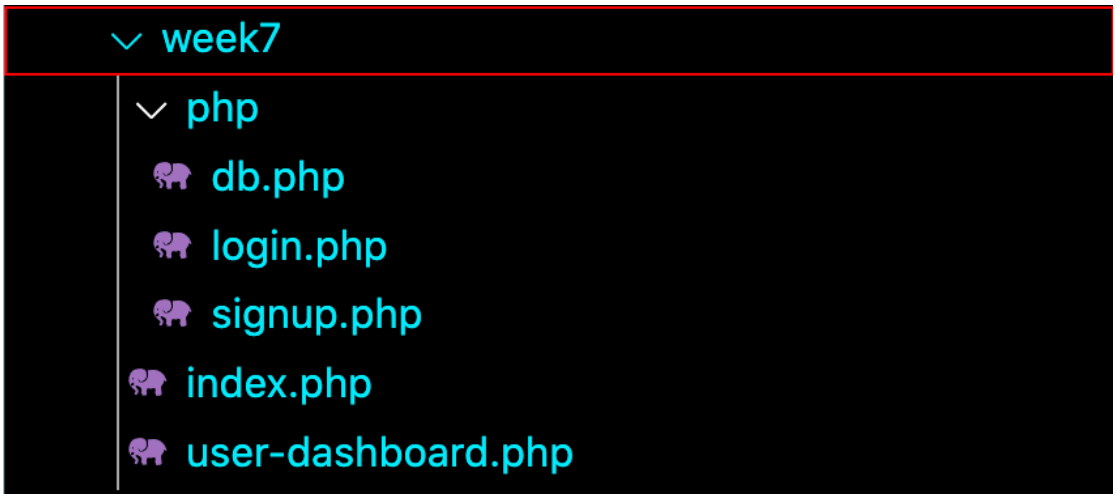
The File Structure

Again there are many different ways we can formulate something like this but in this case, we will have:

index.php - file (this could also be a .html file as it contains no PHP in this case).
user-dashboard.php - file as an example of

where the logged in user could be sent after logging in.
A php subfolder where we will store php files

- db.php - where the database login credentials will be stored we can use `include_once 'db.php';` in any file we may need it.
- login.php - this is the file that will handle the login process.
- signup.php - this is the file that will handle the sign up process.



The HTML

As you would guess we need to have a form the user can use to login on the front end. This can be done in many ways from a design

standpoint. Either way we will need the forms and direct the forms to the files to handle the sign up and login processes using the POST method.

```
<div class="login">
            <h1>Login</h1>

            <form action="php/login.php"
method="post">
                <label for="login">Login</
label>
                <br>
                <input type="text"
id="login" name="user_email">

                <br>

                <label
for="pass">Password</label>
                <br>
                <input type="password"
id="pass" name="password">

                <br>

                <input type="submit"
value="Login">
            </form>
        </div>
```

```
        <div class="signup">
            <h1>Signup</h1>

            <form action="php/signup.php"
method="post">
                <label for="email">Email</
label>
                <br>
                <input type="email"
id="email" name="user_email">

                <br>

                <label for="usrname">User
Name</label>
                <br>
                <input type="text"
id="usrname" name="user_name">

                <br>

                <label
for="pass">Password</label>
                <br>
                <input type="text"
id="pass" name="password">

                <br>

                <input type="submit">
```

```
        </form>
    </div>
```

Note that in this example we have a user name field. However we will be using the email as the login username. We can still allow the user to enter a user name though, we can use it to personally address them.

The PHP

First off, we have to connect to the database, we have done this before so it will be similar:

db.php

```php
<?php

//DB cred
$hostname = '';
$username = '';
$password = '';
$dbname = '';

//connect
$con = new mysqli(
    $hostname,
    $username,
```

```
    $password,
    $dbname
);

//check connection
if ($con === false) {
    die("ERROR: Could not connect. " .
$mysqli->connect_error);
}
```

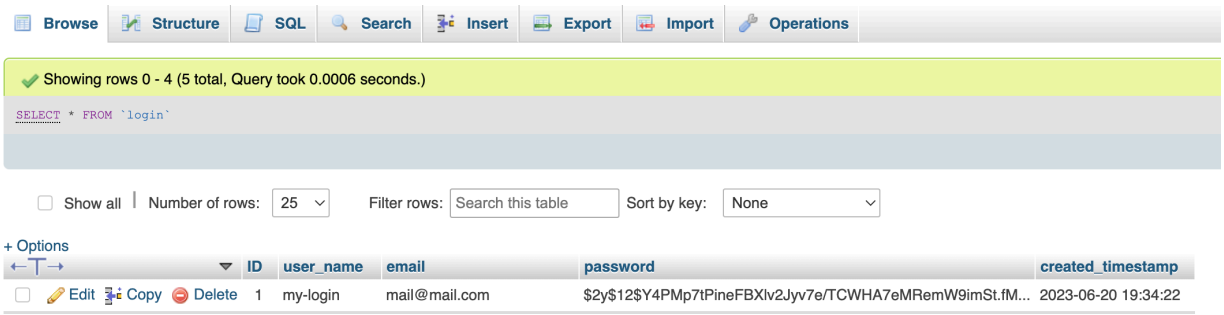
signup.php

```
<?php

//include the DB login
include_once 'db.php';

//gather the form data
$email = $_POST['user_email'];
$username = $_POST['user_name'];
$password =  $_POST['password'];


/*
//Hash the password using Argon2 NOT
WORKING ON INFINITY FREE
$options = [
    'memory_cost' => 1 << 17, // 128MB
    'time_cost'   => 4,
```

```
    'threads'     => 2,
];
$password_encrypt =
password_hash($password, PASSWORD_ARGON2ID,
$options);
*/

//hash the password using Bcrypt
$options = [
    'cost' => 12,//the higher the number
here the more secure however it will also
be more taxing on the server.
];
$password_encrypt =
password_hash($password, PASSWORD_BCRYPT,
$options);

//make a variable and load the SQL
connection and SQL query into it
$stmt = $con->prepare("INSERT INTO login
(user_name, email, password,
created_timestamp) VALUES (?, ?, ?,
current_timestamp())");
if ($stmt === false) {
    die('Error: Failed to prepare the
query.');
}

//bind variables to the table columns that
the corresponding data will be entered into
$stmt->bind_param("sss", $username, $email,
```

```
$password_encrypt);

//execute the query
if ($stmt->execute()) {
    echo 'Form sent!' . '<br>' . '<a
href="../index.php">Back</a>';
} else {
    echo 'Error: Failed to insert data into
the database.';
}

//close all connections
$stmt->close();
$con->close();
```

Assuming that the database is correct, you should see entries populate the DB now. Passwords will be salted and hashed.



login.php

```
<?php
```

```php
// Include the DB login
include_once 'db.php';

// Gather the form data
$email = $_POST['user_email'];
$password = $_POST['password'];

// Prepare and execute the SQL query to
retrieve the user's data
$stmt = $con->prepare("SELECT * FROM login
WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$result = $stmt->get_result();

// Check if the user exists
if ($result->num_rows === 1) {
    $row = $result->fetch_assoc();
    $hashedPassword = $row['password'];

    // Verify the entered password against
the stored hash
    if (password_verify($password,
$hashedPassword)) {

        // Password is correct, login
successful
        echo "Login successful!";

        // Start the session
```

```
            session_start();

            //redirect the user to the
dashboard page

        } else {
            // Password is incorrect
            echo "Invalid password!";
        }
} else {
        // User does not exist
        echo "User does not exist!";
}

//close connections
$stmt->close();
$con->close();
```

Finally:

You can use the `session_start();` On other user pages to make sure they are logged in.