Hello everyone!

Homework

As of last week, we had completed the core of our MVC blog. Through forms and foreach loops in the views, you could Create, Read, Update and Delete bloggers and blog posts. The models defined how to build the SELECT, INSERT, UPDATE, and DELETE sql queries that interacted with the database. And all the controllers sent data from the models to the views or from forms in the views to the models. The only large component missing was security.

The homework this week is to

1. Add logic in the blogger model, controller and view to allow for scrambling passwords (the md5 function).
2. Add logic to the blogger model, controller and view for logging in (setting the $_SESSION cookie).
3. Add logic to the blogger controller and view for logging out (unsetting the $_SESSION cookie).
4. Add logic to the blogger controller to only let the blogger edit her/himself and no one else.
5. Add logic to the post controller to only let people who are logged in create posts, and let the creator of the post edit or delete that post.
6. Add if/else logic to the views to show buttons depending on whether a user is logged in or not.

Class Review

Scrambling Passwords

There are multiple ways of scrambling a password. This is necessary because we don't want our passwords sitting in someone's database where anyone can see them, scrambling them turns normal words like "giraffe" to a scrambled string of letters and numbers that is 64 characters long!

md5 is a very commonly used scramble function in php. There are some downsides, such as, once "giraffe" is scrambled to something like "hfsadjfsadulia388kjhafs98743jkh845iyufahjk98" and someone ELSE uses "giraffe" as their password, it will also be "hfsadjfsadulia388kjhafs98743jkh845iyufahjk98". That means, once someone discovers the scrambled version of a word, all the people who happen to have the same password will be vulnerable to being hacked to.
A more common practice is to use a "salt". A "salt" is a random string of letters and numbers. Then you scramble your password according to an algorithm based off this "salt". That means "giraffe" will scramble to be a different set of 64 characters every time. This is much more secure, but more complicated, so we are going to have to settle for good ol' md5.

In models/blogger.php, you will change two lines to your create() function.

The first line scrambles the password that was put in the ‘password’ field in the view and sent to the model by the controller.
```
$password = md5($fields['password'], false);
```

The second will insert the scrambled password, along with the other data sent by the user into the database.
```
$sql = 'INSERT INTO bloggers (username, email, password,
date_created)VALUES ("' . $fields['username'] . '", "' .
$fields['email'] . '", "'. $password .'", "'. $date .'")';
```

NOTA BENE: you will have to do something very similar for the update() function in your blogger model.

In controllers/blogger.php, you will change two lines in the code that deals with the update_blogger form in the view. (Look for the lines---`if (isset ($_POST['update_blogger']) )`)

Because we scrambled the password in the database when the user was created, we now have to come up with a way to compare the password that the user said was their old password to the scrambled password in the database. Obviously, we can’t expect the user to enter in a scrambled version of their password! So, we scramble what they put in the form and then compare it to what is stored in the database.
```
$blogger = Blogger::getOne($_POST['id']);
     $old_password = md5($_POST['old_password'], false);
     if($old_password == $blogger['password']){
          Blogger::edit($_POST, $_POST['id']);
}
```

Logging in
So far we have created a way for a new blogger to register (create), but we don’t have a way for a blogger to log in. Logging in involves the special variable $_SESSION. This sets cookies for as long as the user is on your site. Despite their bad rap, cookies are very important. Without them, every time you log in into Amazon and then click on a link, Amazon would ask you to log in again. And that would make every site that has a login practically impossible to use. $_SESSION is like the cookie jar. It holds all sorts of different cookies. We will only make one cookie for this application.

First, we add a new function login() to the blogger/model.

Start our function:
```
public static function login ($fields){
```

Clean up the data that the user put into the login form in the view.
```
   $fields = Model::cleanData($fields);
```

Scramble the password that the user gave us:

```
 $password = md5($fields['password'], false);
```

Look in the database for a blogger with the username the user gave us and the scrambled version of the password the user gave us:

```
  $sql = 'SELECT * FROM bloggers WHERE username = "'.
$fields['username'] . '" and password = "' .$password. '"
LIMIT 1';
  $results = Model::select($sql);
```

We have two options. Either there was a blogger with that username and password, and we return data about that blogger. Or there was no one with a matching username and password and we return 'false'.

```
  if($results){
     return $results[0];
  }
  else{
     return false;
  }
}
```

Then, we create the controller code that sends the data from the form in the view to our new model function

We check to see if the login_blogger form was the one sent

```
if(isset($_POST['login_blogger'])){
```

Check that both fields were filled in

```
   if($_POST['username']!='' && $_POST['password']!=''){
```

Send the data to our model function

```
  $blogger = Blogger::login($_POST);
```

If there was a blogger with that username and password, create a cookie named 'user_id' and put it into the $_SESSION cookie jar. That cookie will have the value of the id of the blogger that just logged in.

```
      if($blogger){
         $_SESSION['user_id'] = $blogger['id'];
      }
```

If we didn't get data back from the model, give the user a warning.

```
      else{
         $warning = 'No blogger with that username and
database exists in our database';
      }
   }
```

If one of the fields was empty, send the user a warning.
```
else{
   $warning = 'Please enter both username and password';
}
}
```

Finally, create a form in the view that lets the user login. (See slide 6)

Logging out

Because logging in created a cookie and put it into the $_SESSION cookie jar, logging out will take that cookie out of the cookie jar. Because this doesn't affect the database at all, the model will not change.

First, we create code in the controller that will take the cookie out
We check to see if the logout_blogger form was the one sent
```
if(isset($_POST['logout_blogger'])){
```

Then we use the function "unset" to take out the user_id cookie from $_SESSION
```
   unset($_SESSION['user_id']);
}
```

Then we add a form to the view that lets the user logout (See slide 7)

Using the $_SESSION['user_id']

Now that someone is logged in, we should add some logic to our blogger and post controllers that don't let people edit, delete, or even create things they aren't allowed to.

In the blogger controller, we want to make sure that only the logged in blogger can edit her/himself. So, we add the following line to the code for update_blogger and delete_blogger.
```
if(isset($_SESSION['user_id']) &&
$_SESSION['user_id']==$_POST['id']){
```
This checks first "Is there a cookie named user_id?". Then it checks "Does the user_id held in the cookie match the id of the blogger someone is trying to edit or delete?"

In the posts controller, we want to make sure that only bloggers who are logged in can create posts. We also want to make sure that the only blogger that can edit a post is the author.

For create_post, we check if there is a cookie called user_id, if there is we set the user_id of the blog post to the id of the logged in user.
```
if(isset($_SESSION['user_id'])){
     $_POST['user_id'] = $_SESSION['user_id'];
```

For update_post and delete_post, we check if there is a cookie called user_id and if that cookie matches the user_id of the blog post author.

```
if(isset($_SESSION['user_id']) &&
$_SESSION['user_id']==$_POST['user_id']){
```

If/else logic

There are a lot of buttons visible in the current site. But some of them don't make sense. If I am not logged in, why would I even see buttons for editing a blog post? Also, if I am logged in, why would I even see buttons for editing a blog post that isn't mine? So, in the view, you can add if else logic to show or not show different elements on the page. The example on slide 12 says IF the user is not logged in (there is no cookie named user_id), show a login button ELSE show the logout button.

Use this as a starting point for showing or not showing buttons that a user is allowed to interact with.