Lesson Number 11

Name:

Authentication

Description:

QUIZ - PHP ActiveRecord Recap

Required Knowledge

PHP Active Record/Models

1. What is PHP ActiveRecord?
    - It is an ORM (object relational mapping) Framework based on the active record pattern.
    - It converts the database tables into objects known as "models".
2. What are "models"?
    - Models are objects that centralize our business logic, including associations/relationships, validation, sanitization, and format.
3. When using PHP ActiveRecord, database tables must be in what grammatical form?
    - Plural, as they represent all of the records.
4. The database table must contain a unique identifying column. What name do we give this column as per PHP ActiveRecord?
    - "id" in lowercase format.
5. When designing our database, it is common convention to name our columns as general as possible, without prefixes. ie: name, first_name, last_name, age... Not city_name, person_first_name, person_last_name, dinosaur_age.
    - True as prefixing column names is considered bad practice.
6. Model class names must be in what grammatical form?
    - Singular as they represent a single record.
7. If we have two tables with a parent child relationship, what association do we define in the parent model class?
    - static $has_many = array('children')
    - The programming sentence reads "Parent has many children".
8. What association would we define in the child model class?
    - static $belongs_to = array('parent')
    - The programming sentence reads "Child belongs to parent".
9. In Object Oriented PHP, you will likely use the construct "$this". What does "$this" refer to?
    - $this is a reference to the current object.
10. Setters are defined in the model class and are used to perform mutations on data before we save the data to the database.
    - True. These constructs are great for sanitizing data before we insert or update any records.
11. What is the naming convention for defining a setter in a model class?
    - Use the prefix "set_" then the name of the attribute (database column) you wish to affect. ie: set_name, set_first_name, set_last_name, set_age
12. In a setter, in order to assign the new value, what method must we use on $this, and what arguments does it require?
    - assign_attribute('attribute name', 'value we wish to assign'). This method is part of the extended class ActiveRecord\Model.

13. Getters are defined in the model class and are used to perform mutations on data before output.
    - True. These constructs are great for sanitizing data before we read any records.
14. What is the naming convention for defining a getter in a model class?
    - Use the prefix "get_" then the name of the attribute (database column) you wish to affect. ie: get_name, get_first_name, get_last_name, get_age
15. In a Getter, in order to manipulate the data for output, we must use what method on $this, and what argument does it require?
    - read_attribute('attribute name'). This method is part of the extended class ActiveRecord\Model.
16. Getters and Setters, defined with existing attribute names, are called automatically when creating, reading, updating, or deleting records.
    - True. PHP ActiveRecord will automatically call these methods, hence why they are so powerful and convenient.
17. Getters defined with unique names, are called manually by the developer.
    - True. These are great for formatting data in a specific format for output. ie: prices, full name, current exchange rate, etc...
18. When validating in a model, PHP ActiveRecord gives us a number of predefined validators. Give an example of one of these validators.
    - static $validates_presence_of = array(array('name', 'message' => 'must be present.'))
19. How do you create a custom validator in PHP ActiveRecord?
    - By defining a public function with the label "validate".

Views

1. What is a view?
    - A view is the interface for our user. It displays data sent from our controller, and receives input from the user.

Controller

1. What is the purpose of the controller?
    - The controller works as a bridge between our models and our views.
    - It processes data from the model and provides it to the view.
    - It centralizes our CRUD logic.
2. What are common functions defined in the controller and what do they **usually** represent?
    - index: a view of allrecords from a resource
    - show: a view of one record from a resource
    - create: a view for user input to create a new resource
    - edit: a view for a user to change input for an existing resource
    - add: processes the **posted** form data from the **create** page and creates a new record
    - edit: processes the **posted** form data from the **edit** page and updates an existing record
    - delete: processes the **posted** request and deletes an existing record
3. How does the controller provide the view to the index page?

Action Handler

1. What is the function of the action handler?
    - The action handler works by calling the requested action from a resource's controller.
    - The requested action is a keyword that reflects a defined function in the controller.

Resource Requirements

1. When adding a new resource to our application what requirements must we have?
    - A named folder for the resource (usually named after the database table) ie: categories,

users, products
    - A contained folder titled "views"
    - "views" will contain files to be displayed to the user for user output/input.
    - A controller file that will contain actions/functions for the resource
    - An index page that will act as our final output for the user.

Application Flow

1. What is the application flow to view all the categories in our application?
    1. The request: **/categories/index.php?action=index** is sent
    2. The request is intercepted by the **action handler**
    3. The action handler verifies the requested action **"index"** exists in the controller
    4. The function **index()** is called in the **controller**
    5. The model is queried for all the categories and the result is stored in a variable called **$categories**
    6. The requested view is included and returned to the handler
    7. The handler stores the result in a variable called **$yield**
    8. The index page then outputs the contents of **$yield**
2. How does **get_included_file_contents($path, $params = [])** work?
    - On occasion, you may need to create dynamic variables. These allow you to create a variable label dynamically and store a value in it.
    - In get_included_file_contents, you will see the double $$. This represents a dynamic variable, also known as a variable variable.
    - We utilize these in get_included_file_contents so we can pass in parameters to the included file. These get passed in an associative array containing a list of keys and values. Each key represents the name of the variable where as the value will be the value to be stored when the variable is created.
    - In PHP, there a few ways to parse the PHP in a file. The easiest way is to use one of the four include functions.
    - The include functions will immediately output the content requested. Sometimes you may want to store that parsed output instead of immediately displaying it.
    - PHP has a few functions that allow you temporarily store any output in a buffer which restricts it from displaying.
    - ob_start() will collect any output and store it.
    - ob_get_contents() will return the current contents in the buffer
    - ob_end_clean() will clear the buffers contents.
    - In get_included_file_contents, we start the buffer, store the contents in a variable, then clear the buffer. Once that is finished, we return the buffer.

The Action Handler

The Action Handler

ACTIVITY - Users

Adding the User Resource

```
CREATE TABLE `users` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `first_name` varchar(50) NOT NULL,
  `last_name` varchar(50) NOT NULL,
  `email` varchar(50) NOT NULL,
  `password` varchar(100) NOT NULL,
  `role` tinyint(4) NOT NULL DEFAULT '1',
  PRIMARY KEY (`id`)
)
```

1. Add the SQL above to the database that contains categories and products
2. Add a **users** folder to the project
3. Add the following files to the **users** folder
    1. index.php
    2. controller.php
4. Add a **views** folder to the **users** folder
5. Add the following files to the **views** folder:
    1. index.php
    2. form.php
    3. create.php
    4. edit.php

ACTIVITIY - users/index.php

```
<?php

  // require the controller
  require_once 'controller.php';

  // require the HTML header
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/includes/header.php';

  // output
  echo $yield;


  // require the HTML footer
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/includes/footer.php';
```

ACTIVITY - controller.php

```
<?php

  // start our session to avoid headers issue
  session_start();

  /* ACTION HANDLER */
  // attach PHP ActiveRecord
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/config.php';


  /* VIEWS */
  function index () {
    $users = User::all( array( 'order' => 'last_name' ) );
    return get_included_file_contents( 'views/index.php',
['users' => $users] );
  }

  function create () {
    return get_included_file_contents( 'views/create.php' );
  }

  function edit ( $get ) {
    if ( !isset( $get['id'] ) || !User::exists( $get['id'] ) )
{
      $_SESSION['fail'] = "You must select a category.";
      header( 'Location: index.php?action=index' );
      exit;
    }

    $user = User::find( 'first', $get['id'] );
    return get_included_file_contents( 'views/edit.php',
['user' => $user] );
  }


  /* PROCESSES */
  function add ( $post ) {
    // create a new record
    $user = new User;

    // assign the values
    $user->first_name = $post['first_name'];
    $user->last_name = $post['last_name'];
    $user->email = $post['email'];

    // confirm the passwords match
```

```
                if ( $post['password'] == $post['confirm_password'] ) {
                  // hash the password
                  $user->password = password_hash( $post['password'],
PASSWORD_DEFAULT );

                  // set the confirm password to the new hashed password
so it passes validation
                  $user->confirm_password = $user->password;
                } else {
                  // set the password to the current post password
                  $user->password = $post['password'];

                  // set the confirm password so it fails the compare
validation
                  $user->confirm_password = null;
                }

                // when we save, we apply our assigned properties and
write them to the database
                $user->save();

                // redirect if there is an error
                if ( $user->is_invalid() ) {
                  // set the fail messages
                  $_SESSION['fail'][] = $user->errors->full_messages();
                  $_SESSION['fail'][] = 'The user could not be created.';

                  // redirect
                  header( 'Location: index.php?action=create' );
                  exit;
                }

                // set the success message
                $_SESSION['success'] = 'User was created successfully.';
                header( 'Location: ../authentication/index.php?
action=login' );
                exit;
              }

              function update ( $post ) {
                // redirect user if here accidentally
                if ( !isset( $post['id'] ) || !User::exists( $post['id'] )
) {
                  $_SESSION['fail'] = "You must select a user.";
                  header( 'Location: index.php?action=index' );
                  exit;
                }

                // get existing record
                $user = User::find( $post['id'] );
```

```
                    // update the values
                    $user->first_name = $post['first_name'];
                    $user->last_name = $post['last_name'];
                    $user->email = $post['email'];

                    // if not empty, update
                    if ( !empty( $post['password'] ) ) {
                      // confirm the passwords match
                      if ( $post['password'] == $post['confirm_password'] ) {
                        // hash the password
                        $user->password = password_hash( $post['password'],
PASSWORD_DEFAULT );

                        // set the confirm password to the new hashed password
so it passes validation
                        $user->confirm_password = $user->password;
                      } else {
                        // set the password to the current post password
                        $user->password = $post['password'];

                        // set the confirm password so it fails the compare
validation
                        $user->confirm_password = null;
                      }
                    }

                    // when we save, we apply our assigned properties and
update them in the database
                    $user->save();

                    // redirect if there is an error
                    if ( $user->is_invalid() ) {
                      // set the fail messages
                      $_SESSION['fail'][] = $user->errors->full_messages();
                      $_SESSION['fail'][] = 'The user could not be updated.';

                      // redirect
                      header( 'Location: index.php?action=edit&id=' .
$post['id'] );
                      exit;
                    }

                    // set the success message
                    $_SESSION['success'] = 'User was updated successfully.';
                    header( 'Location: index.php?action=index' );
                    exit;
                  }

                  function delete ( $post ) {
                    // redirect user if here accidentally
                    if ( !isset( $post['id'] ) || !User::exists( $post['id'] )
```

```
) {
                    $_SESSION['fail'] = "You must select a category.";
                    header( 'Location: index.php?action=index' );
                    exit;
                  }

                  // delete the record
                  $category = User::find( $post['id'] );
                  $category->delete();

                  $_SESSION['success'] = 'The user was deleted
successfully.';
                  header( 'Location: index.php?action=index' );
                  exit;
                }


                /* Authentication */
                request_is_authenticated( $_REQUEST, ['create', 'add'] );


                // action handler for REQUEST
                $yield = action_handler( ['add', 'update', 'delete',
'index', 'create', 'edit'], $_REQUEST );
```

ACTIVITY - users/views/index.php

```
            <div class="container">
              <h1 class="page-header">Users</h1>
              <p><a href="?action=create"><i class="fa fa-
plus"> </i>Create User</a></p>

              <?php if ( isset( $users ) ): ?>
                <table class="table table-striped table-condensed table-
hover">
                  <thead>
                    <tr>
                      <th>First Name</th>
                      <th>Last Name</th>
                      <th>Email</th>
                      <th>Show</th>
                      <th>Edit</th>
                      <th>Delete</th>
                    </tr>
                  </thead>
```

```
                    <tbody>
                      <?php foreach ( $users as $user ): ?>
                        <tr>
                          <td><?= $user->first_name ?></td>
                          <td><?= $user->last_name ?></td>
                          <td><?= $user->email ?></td>
                          <td><a href="?action=show&id=<?= $user->id ?>"><i
class="fa fa-eye"></i></a></td>
                          <td><a href="?action=edit&id=<?= $user->id ?>"><i
class="fa fa-pencil"></i></a></td>
                          <td>
                            <form action="controller.php" method="post">
                              <input type="hidden" name="action"
value="delete">
                              <input type="hidden" name="id" value="<?=
$user->id ?>">
                              <button type="submit" style="border: none;
background: none; color: #337ab7; padding: 0; margin: 0;" onclick="return
confirm('Are you sure you want to permanently delete <?= $user->first_name .
' ' . $user->last_name ?>')">
                                <i class="fa fa-remove"></i>
                              </button>
                            </form>
                          </td>
                        </tr>
                      <?php endforeach ?>
                    </tbody>
                  </table>
                <?php endif ?>
              </div>
```

ACTIVITY - users/views/form.php

```
              <form action="controller.php" method="post">

                <fieldset>
                  <legend>User Information</legend>

                  <div class="form-group">
                    <label for="first_name">First Name</label>
                    <input class="form-control" type="text"
name="first_name" value="<?= isset( $user ) ? $user->first_name : '' ?>"
required maxlength="100">
                  </div>
```

```
                <div class="form-group">
                  <label for="last_name">Last Name</label>
                  <input class="form-control" type="text" name="last_name"
value="<?= isset( $user ) ? $user->last_name : '' ?>" required
maxlength="100">
                </div>

                <div class="form-group">
                  <label for="email">Email</label>
                  <input class="form-control" type="text" name="email"
value="<?= isset( $user ) ? $user->email : '' ?>" required maxlength="100">
                </div>

                <div class="form-group">
                  <label for="password">Password</label>
                  <input class="form-control" type="password"
name="password" <?= isset( $action ) && $action == 'update' ? '' :
'required' ?> maxlength="100" minlength="8">
                </div>

                <div class="form-group">
                  <label for="confirm_password">Confirm Password</label>
                  <input class="form-control" type="password"
name="confirm_password" <?= isset( $action ) && $action == 'update' ? '' :
'required' ?> maxlength="100" minlength="8">
                </div>

                <div class="form-group">
                  <input type="hidden" name="action" value="<?= isset(
$action ) ? $action : 'add' ?>">

                  <?php if ( isset( $action ) && $action == 'update' ): ?>
                    <input type="hidden" name="id" value="<?= $user->id ?
>">
                    <button type="submit" class="btn btn-danger"><i
class="fa fa-pencil"> </i>Update User</button>
                  <?php else: ?>
                    <button type="submit" class="btn btn-danger"><i
class="fa fa-plus"> </i>Add User</button>
                  <?php endif ?>
                </div>
              </fieldset>

            </form>
```

ACTIVITY - users/views/create.php & users/views/edit.php

create.php

```
<div class="container">
  <h1 class="page-header">Create User</h1>

  <?php include 'form.php' ?>
</div>
```

edit.php

```
<div class="container">
  <h1 class="page-header">Edit User</h1>

  <?php $action = 'update' ?>
  <?php include 'form.php' ?>
</div>
```

Authentication

Authentication Explained

- In programming you will likely require authentication
- Authentication protects the following examples:
  - User/Customer details
  - Business data
  - Payment information
  - API data
  - File Access
  - OS Access
  - Social media interactions
- The following authentications are common
  - Basic Authentication
    - this is serverside authentication
    - can be created in varying ways including an htaccess file
    - requires a username and a password
    - base64 encodes the password
    - only secure in an HTTPS environment
  - Session Authentication
    - this is serverside authentication
    - created by storing username and password in a datastore
    - this authentication requires the developer to encrypt/hash user passwords and establish rules for passwords to protect against attacks
    - subject to brute-force attacks if the developer hasn't created logic to disable the

account upon detection
    - OAuth
        - Open Authorization protocol
        - allows applications to authenticate as users
        - requires a generated authentication token to work
        - authentication tokens are usually awarded after successful submission of secret key and secret code
        - authentication tokens will often expire after a period of time or if a different referral IP is making the request
    - OpenID & SAML
        - OpenId is an HTTP-based protocol that uses identity providers to validate a user
        - SAML is like OpenId but utilizes XML
        - Both are considered Single-Sign-On (SSO) authentication methods
        - SSO allows for a user to access several websites without the need to reauthenticate
    - Two-Factor Authentication
        - This authentication protocol requires a user to sign in with a username and password, then enter a provided key usually sent through email or SMS message.
        - Two-factor authentication is powerful as it ensure the identity of the user and is almost impossible to circumvent
- Rules for authentication: https://www.owasp.org/index.php/Authentication_Cheat_Sheet
    - Use case sensitive and unique user IDs or names. Email addresses are ususally best.
    - Enforce a minumum password length.
    - Enforce password complexity.
    - Disallow common password patterns or phrases.
    - Implement a secure password recovery mechanism.
    - Store passwords in a single direction cryptographic fashion. Hashing.
    - Transmit passwords over TLS or SSL only.
    - If storing authentication for quick login, re-authenticate for sensitive features such as profile or password changes.
    - Practice security by obscurity methods. Be vague in login error messages. Avoid the following phrases:
        - Invalid password
        - Invalid user
        - Account disabled
        - User not active
- Use a respectable authentication library

ACTIVITY - Authentication

Adding Authentication Module

1. Add a **authentication** folder to the project
2. Add the following files to the **authentication** folder
    1. index.php
    2. controller.php
3. Add a **views** folder to the **authentication** folder
4. Add the following files to the **views** folder:
    1. login.php

ACTIVITIY - config.php - request_is_authenticated($request, $whitelist)

Request is Authenticated

ACTIVITY - authentication/index.php

```
<?php

  // require the controller
  require_once 'controller.php';

  // require the HTML header
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/includes/header.php';

  // output
  echo $yield;


  // require the HTML footer
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/includes/footer.php';
```

ACTIVITY - authentication/controller.php

```
<?php

  // start our session to avoid headers issue
  session_start();

  /* ACTION HANDLER */
  // attach PHP ActiveRecord
  require_once $_SERVER['DOCUMENT_ROOT'] . '/lesson-
11/examples/config.php';

  /* VIEWS */
  function login () {
    return get_included_file_contents( 'views/login.php' );
```

```
}


/* PROCESSES */
function authenticate ( $post ) {
  $user = User::find( 'first', array( 'email' =>
$post['email'] ) );
  if ( $user && password_verify( $post['password'], $user-
>password ) ) {
    $_SESSION['success'] = 'You have successfully logged
in.';
    $_SESSION['authenticated'] = true;
    $_SESSION['email'] = $user->email;
    header( 'Location: ../categories/index.php?action=index'
);
    exit;
  } else {
    $_SESSION['fail'] = 'You could not be logged in at this
time.';
    header( 'Location: index.php?action=login' );
    exit;
  }
}

function logout () {
  if ( isset( $_SESSION['authenticated'] ) ) {
    unset( $_SESSION['authenticated'] );
    unset( $_SESSION['email'] );
    $_SESSION['success'] = 'You have been logged out
successfully.';
    header( 'Location: index.php?action=login' );
    exit;
  }
}

/* Authentication */
request_is_authenticated( $_REQUEST, ['login',
'authenticate'] );


// action handler for REQUEST
$yield = action_handler( ['login', 'logout',
'authenticate'], $_REQUEST );
```

ACTIVITY - authentication/views/login.php

```
            <div class="container">
              <h1 class="page-header">Login</h1>

              <form action="controller.php" method="post">
                <fieldset>
                  <legend>Login</legend>

                  <div class="form-group">
                    <label for="email">Email</label>
                    <input class="form-control" type="email" name="email"
required maxlength="100">
                  </div>

                  <div class="form-group">
                    <label for="password">Password</label>
                    <input class="form-control" type="password"
name="password" required maxlength="100" minlength="8">
                  </div>

                  <div class="form-group">
                    <input type="hidden" name="action"
value="authenticate">
                    <button type="submit" class="btn btn-danger"><i
class="fa fa-sign-in"> </i>Login</button>
                  </div>
                </fieldset>
              </form>
            </div>
```