Lesson 5

# PHP, Forms and Databases

PHP only works on a live server. This is where software such as MAMP will come in handy as you it will allow us to test PHP on a local server or you can use FTP (Cyber duck or another program) to publish to a remote server.

### Sending an email with JavaScript AJAX and PHP

|  |  |
| --- | --- |
| JavaScript AJAX  First we’ll set up the javaScript which will handle our HTML form data.  Copy the following code into the mail.js file.  $(function() {  // Get the form with an ID of “ajax-contact” .  var form = $('#ajax-contact');  // Get the messages div with an ID of “form-messages”.  var formMessages = $('#form-messages');  // TODO: The rest of the code will go here...  });  Here you’ve created two new variables, form and formMessages, that reference the corresponding elements in your HTML markup. | |
| Create an event listener that will intercept form submit events.  // Set up an event listener for the contact form.  $(form).submit(function(event) {  // Stop the browser from submitting the form.  event.preventDefault();  // TODO  });  Here you’ve passed a function to the jQuery submit method that will be executed when the user submits the contact form.  The preventDefault method tells the form not to behave normally as we’re going to give it different instructions. Normally the page would redirect to another page, but using JavaScript AJAX we are just telling it to use the php file instead of viewing it, this will make sure we stay on the same page. | |
| Convert the data the user has entered into a key and value string that can be sent to php.  An example of a key is:  Name(Key): Joe(Value).  // Serialize the form data.  var formData = $(form).serialize();  We use the jQuery serialize method to serialize the form data and then store the result in a variable called formData. | |
| Send the form data to the server and processes the response.  // Submit the form using AJAX.  $.ajax({  type: 'POST',  url: $(form).attr('action'),  data: formData  })  Using jQuery’s ajax method we pass an object that contains properties used to configure the request.  Type: specifies the HTTP method that will be used for the request.  Url: is the location of the script that the form data will be sent to, which is (in this case) taken from the form’s ‘action’ attribute.  Data: use the formData variable that we created earlier. | |
| Next we handle a successful response from the server.  Copy the following code directly after the closing bracket of the ajax call. Note that we’ve deliberately left out the semicolon at the end.  .done(function(response) {  // Make sure that formMessages has the 'success' class.  $(formMessages).removeClass('error');  $(formMessages).addClass('success');  // Set the message text.  $(formMessages).text(response);  // Clear the form.  $('#name').val('');  $('#email').val('');  $('#message').val('');  })  This “done” method will be called if the request completes successfully.  First make sure that the formMessages element has the success class and then set the text content of this element using the data returned by the mailer script.  Then, we reset the form fields. | |
| The last bit of JavaScript handles what should happen if an error occurs.  .fail(function(data) {  // Make sure that formMessages has the 'error' class.  $(formMessages).removeClass('success');  $(formMessages).addClass('error');  // Set the message text.  if (data.responseText !== '') {  $(formMessages).text(data.responseText);  } else {  //This fail method is called if the mailer script returns an error.  $(formMessages).text('Oops! An error occurred and your message could not be sent.');  }  });  We add an error style to the formMessage.  Check to see if the AJAX request returned a response and set our html form message to display the response, and if it doesn’t have a response create a response message. | |

### 

|  |  |
| --- | --- |
| PHP mailer  Now we’ll write the PHP mailer script that will process the form data.  This simple script is responsible for checking that the form data is valid and then sends the email.  Open mail.php and add the following:  <?php  // Only process POST requests.  if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  // Get the form fields and remove whitespace.  $name = strip\_tags(trim($\_POST["name"]));  $name = str\_replace(array("\r","\n"),array(" "," "),$name);  $email = filter\_var(trim($\_POST["email"]), FILTER\_SANITIZE\_EMAIL);  $message = trim($\_POST["message"]);  // Check that data was sent to the mailer.  if ( empty($name) OR empty($message) OR !filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  // Set a 400 (bad request) response code and exit.  http\_response\_code(400);  echo "Oops! There was a problem with your submission. Please complete the form and try again.";  exit;  }  // Set the recipient email address.  // FIXME: Update this to your desired email address.  $recipient = "hello@example.com";  // Set the email subject.  $subject = "New contact from $name";  Build the email content.  $email\_content = "Name: $name\n";  $email\_content .= "Email: $email\n\n";  $email\_content .= "Message:\n$message\n";  // Build the email headers.  $email\_headers = "From: $name <$email>";  // Send the email.  if (mail($recipient, $subject, $email\_content, $email\_headers)) {  // Set a 200 (okay) response code.  http\_response\_code(200);  echo "Thank You! Your message has been sent.";  } else {  // Set a 500 (internal server error) response code.  http\_response\_code(500);  echo "Oops! Something went wrong and we couldn't send your message.";  }  } else {  // Not a POST request, set a 403 (forbidden) response code.  http\_response\_code(403);  echo "There was a problem with your submission, please try again.";  }  ?> | |
| If the request was not sent using the POST method, the script will return a 403 (forbidden) HTTP status code and an error message.  We then extract the form data into three variables $name, $email, and $message. We use the PHP trim method to remove whitespace.  Check that no fields are blank. If one or more is blank, set the response code to 400 (bad request) and return an error message.  Now we’ve determined the data is clean we create variables for the email recipient, subject, email content, and email headers. Then send the mail.  *Setting the email headers is optional. It’s important to note that manipulating the headers can cause the email to be marked as spam by some email clients.*  If the mail is successful return a success message. If it’s not, set the response code to 500 (internal server error) and return an error message. | |
| Publish your pages using FTP (cyberduck) to a server.  Test your form on a live server! | |

### 

### MySQL Databases

|  |  |
| --- | --- |
| Open data.php  First we set up php variables for important data we need to connect to the database, and the data we collect from the form. Enter this:  <?php  $servername = "localhost";  $username = "admin";  $password = "admin";  $dbname = "myDB";  $table = "people";  $fname = $\_POST["fname"];  $lname = $\_POST["lname"];  $age = $\_POST["age"];  $email = $\_POST["email"];  // Todo  ?> | |
| Next add this:  // Create connection  $conn = new mysqli($servername, $username, $password, $dbname);  // Check connection  if ($conn->connect\_error) {  die("Connection failed: " . $conn->connect\_error);  }  Here we create a connection, and check if there is an error. | |
| Next we post our data to the database  $sql = "INSERT INTO $table (fname, lname, age, email) VALUES ('$fname', '$lname', '$age', '$email')"; | |
| Finally add this:  if ($conn->query($sql) === TRUE) {  echo "New record created successfully";  } else {  echo "Error: " . $sql . "<br>" . $conn->error;  }  $conn->close();  Here we respond if the connection was successful, or return an error if it fails, then close the connection. | |
| Open the page on the server.  Test your form, check phpmyadmin that the data has been passed into the database.  That’s it! You’ve added a database entry using php! | |