

5.2: Integration Examples

The following code examples will demonstrate how the languages we have studied can be combined to create dynamic systems. In each of these examples two or more elements covered in the text will be combined. These examples are not intended to function fully based on the excerpts you will see, but are meant to demonstrate methods of integrating the languages.

Connecting to MySQL

In order to have our users interact with our database, we need to establish a bridge of communication between them. To do this, we will create a connection to our database and store it in a variable. Through PHP we have a variety of methods of creating this connection, among them are libraries called MySQL, MySQLi, and PDO. These libraries have different approaches to interacting with a database, and there are differences in what commands are available and how the connection is treated. While you will see many examples online that use the standard MySQL connector, I would warn you away from it. It is on its way to deprecation, and has little injection protection built in. The MySQLi library is the one we will focus on in our examples here as it is a better start for entry level programmers. Ultimately, once you are comfortable with integrating these languages, I would recommend moving to PDO. While it is not tailored for MySQL, it supports a wider range of SQL databases that will allow you to more easily change your backend system.

To begin, we will call a function to create our connection. The shortest avenue to do this is as follows:

```
1. $mysql = mysqli->connect("localhost","user","password","database");
```

By inserting your server's values in each set of quotes, the variable \$mysql will become our line of communication to our MySQL database. When we created our connection by using a class method, our \$mysql variable is now a MySQLi object. We could also have used procedural style with `mysqli_connect`. Assuming your database is the same system as your website, "localhost" or "127.0.0.1" should work just fine for you. Your username and password can be any account that exists in your SQL database. In a fresh installation, "root" as the user and "root," "password," or nothing—" " as a password will usually let you in, but as we saw in security, you should avoid this unless necessary and only on a low-risk machine. The declaration of the database we want to use is optional, but saves us from having to select one later or always declare our database in our queries.

In the spirit of this section, we will revise this example to make it more useful. By replacing our values with variables, we can keep our actual values apart from the rest of the code. Additionally, we can separate our connection out to its own file. By doing this, we can `require()` or `include()` it on any page that we need a connection. Then all we need to do when we use our database is remember to close the connection at the bottom of our page. An additional advantage is that we could also wrap our connection in a class of our own, allowing us to re-declare from our own class what functions are available. For now, we will keep it simpler:

```
1. $host = "localhost";
2. $user = "username";
3. $password = "password";
4. $dbase = "database";
5. $mysql = mysqli->connect($host, $user, $password, $dbase);
```

If this code is its own file like `database.php`, we could use it in all of our sites, simply changing the values at the top to match the settings for the site it is in. To get information from our database, create or modify our databases, or create or change records, we use the same exact queries that we did from the command prompt. Our only difference is that we do it through a function, and store the response in a variable:

```
1. $results = $mysql->query("select * from albums");
```

The \$results variable here, like our connection variable, is a reference. It is a pointer that lets us continue to communicate with the database and specifies what we are looking for, but is not the actual data. To get the data, we need to ask for each record. Since our example is very small, we will get all of the results at once, and build an array:

```
1. while($row = $results->fetch_assoc){
2. $data[]=$row;
3. }
```

This block of code uses the while statement to get every record available in our result set. Note that we used the variable with our result pointer to get the results, and not our connection itself. Inside our loop, we are simply taking the row (in this case, each

album from our albums table) and adding it to a new array called data.

Secured Login

Logging into a web page involves receiving user input, sanitizing and validating their submission, appending any salts, hashing the submission, and sending it to the database for verification. If the database responds that the user's credentials match what is stored we can then continue and create a cookie and/or session so the user can interact with secured content.

Creating a User:

```
1. <?php
2. //create our salt
3. $salt=^%r8yuyg;
4. //store the filtered, salted, hashed version of the password
5. $passwordHash = sha1(filter_var($_POST['password'],$salt, FILTER_SANITIZE_STRING));
6. //Add the user to the database
7. $sql = 'INSERT INTO user ($username, passwordHash) VALUES (?,?)';
8. $result = $db->query($sql, array($_POST['username'], $passwordHash));
9. ?>
```

Logging them in:

```
1. <?php
2. //Prep their login credentials
3. $passwordHash = sha1(filter_var($_POST['password'],$salt, FILTER_SANITIZE_STRING));
4. $sql = 'SELECT username FROM user WHERE username = ? AND passwordHash = ?';
5. $result = $db->query($sql, array($_POST['username'], $passwordHash));
6. //This time, look at the result to see if they exist
7. if ($result->numRows() < 1){
8. echo 'Sorry, your username or password was incorrect!';
9. }
10. else{
11. // Create the session
12. $session_start();
13. $_SESSION['active'] = true;
14. echo ("Welcome back!");
15. }
16. ?>
```

Dynamic Canvas

By adding a loop in a canvas drawing of a circle, and using the random number function in the math library of JavaScript, we can instruct the browser to draw circles (in this example, 1000) of random shapes, sizes, and colors, all over our canvas. Each time the page is loaded or refreshed in the browser, the circles will be redrawn, and since we are using random values, this means our image will change each time.

```
1. <canvas id="myCanvas" width="600" height="600"></canvas>
2. <script>
3. var canvas = document.getElementById("canvas");
4. var ctx = canvas.getContext("2d");
5. var w = canvas.width, h = canvas.height; //Set variables of the width, height of the canvas
6. var i = 0;
7. do {
8. ctx.fillStyle = "rgb(" + Math.round(255*Math.random()) + "," // Creates an R value
9. + Math.round(255*Math.random()) + "," // ... and for G
10. + Math.round(255*Math.random()) + ")"; // ... and for B
11. ctx.beginPath();
```

```
12. ctx.arc(w*Math.random(), h*Math.random(), // creates our random size
13. 50*Math.random(),
14. 0, Math.PI*2, true); // Uses Pi*2 to make the arc a circle
15. ctx.closePath();
16. ctx.fill();
17. } while (++i != 1000); // Loops this do 999 more times
18. </script>
```

Table of Results

If we only need to display the information to the user on the screen, or do not plan on manipulating the data or using the data elsewhere on our page, creating the array of results can be a wasteful use of memory. Instead, we will modify our while loop to create a table of our results:

```
1. <table width="75%">
2. <tr><th>Title</th><th>Artist</th><th>Year</th><tr>
3. <?php
4. while($row = $results->fetch_assoc){
5. echo "<tr><td>$row[title]</td><td>$row[artist]</td><td>$row[year]</td></tr>";
6. }
7. ?>
8. </table>
```

This approach only stores one record at a time in our \$row variable and is much more conservative, especially when using larger data sets. You will notice we nested our PHP within regular html in this example. Take a look at what our whole page might look like, assuming we also created our database.php file:

```
1. <?php
2. require("database.php"); // Now that this file is loaded, we can use $mysql on this page
3. $query = "select title, artist, year from albums";
4. $results = $mysql->query($query);
5. ?>
6. <table width="75%">
7. <tr><th>Title</th><th>Artist</th><th>Year</th><tr>
8. <?php
9. while($row = $results->fetch_assoc){
10. echo "<tr><td>$row[title]</td><td>$row[artist]</td><td>$row[year]</td></tr>";
11. }
12. ?>
13. </table>
```

At this point, we now have all of the data we need from the database, and since we will not need the database anymore for this example, we can close our connection by adding the following after our table:

```
1. <?php mysql->close($mysql); ?>
```

Repopulating Forms

If a user has submitted a form or is editing existing data, we can use these values to repopulate our form so the user does not have to type them in again. This is done by echoing the value inside the quotes in the value attribute of the form element. For example, if our name was in the URI as page.php?name=myName, we could make it the value of the input field with:

```
1. <form action='page.php' method='get'>
2. <input type='text' value='<?php echo $_GET['name']; ?>' />
3. </form>
```

By suppressing errors with error_reporting, using this technique, and with a little logic, we can combine all of the elements of providing the form, validating the submission, and taking action on what the form is for, all from one single page.

With some careful planning, we can provide our user with a form, check their submission, and store their information, all from one page. First we need to create some pseudo-code of the flow of logic so we can dictate what we want to do, and under what conditions:

```
1. <?php
2. if(form has not been submitted){
3. show user the blank form
4. }
5. else{
6. check the form for errors
7. if (there are errors){
8. show user their form and data
9. }
10. }
11. ?>
```

By following the logical order of events, the above pseudo-code represents what would be a perfectly functional page. With a couple of tweaks, however, we can make some improvements. First, in this example, we would have to create the same form twice—once blank, and again with placeholders for what the user submitted. While we could use copy/paste and then modify the second copy, this will greatly inflate the overall size of our page. We can also simplify our logic by reversing the order of events in our code. First, we will see if the form has been submitted. If it has, we will check for errors. If there are none, we will complete the submission. After that, we will simply display the form with placeholders for user submitted data if there are errors or the form has not been submitted. This will cover both cases in one place and replaces our if/else and nested if with three if statements:

```
1. <?php
2. if(form has been submitted){
3. check it for errors.
4. create a status flag declaring if there are errors or not
5. }
6. if(the status flag is set to "ok"){ // The form must have been submitted, and there are no errors
7. submit the user info to the database
8. send any confirmation emails
9. display a success message
10. set the status flag to show the form is complete
11. }
12. if(the status flag is anything other than "ok" or does not exist){ //either there were errors, or the form has not been submitted
13. show the form with placeholders for submitted data
14. }
15. ?>
```

To make this form more flexible, we can declare an array of field names in our first if statement that lists what elements in our table are required or need to be validated. Once we have done this, we can check each submitted field to see if it needs to be checked, pass it through the appropriate tests, and create a list of feedback. There are a number of ways to approach this. Here we will create an array of responses as our error flag. If there are no errors, we will simply set it to “OK.” We will create a hidden field with a known value that will be included every time the form is submitted. This will help deter outside scripts from using our form, as they would need to know to include the hidden field (which we will check for in our logic) in order for our script to respond. In our example, we will make a short registration form:

```
1. if($_GET[hiddenDate]==now() [check]){
2. $check = array('firstName', 'lastName', 'email', 'email2'); //We will require these fields
3. foreach($check as $field){ [make sure & is in php section] [include "for each thing in" way to remember
4. if($field in $_GET) [verify]{
5. //sanitize variable
6. if(length < 3){ //establishes that our required fields should be at least 3 characters long
7. $_GET[$field]=""; //clear the user submitted value as it does not qualify
```

```
8. $errors[]="$field is required and must be at least 3 characters long";
9. }
10. }
11. }
12. if($_GET['email'] != $_GET['email2']){ // Make sure the user entered the same email twice
13. $errors[]="Both email fields must match";
14. $_GET['email']=""; $_GET['email2']="";
15. }
16. else{ // email wasn't entered and/or fields matched
17. if(!empty($_GET['email'])){ // Eliminate the possibility that email is simply empty
18. if(validate($_GET['email'], EMAIL)==false{ $errors[]="Invalid email address" [check]; // We only need to validate one, since
    they are the same
19. $_GET['email']=""; $_GET['email2']="";
20. }
21. }
22. if(!isset($errors)){ // if nothing tripped an error, the array was never created
23. $errors='ok'; // Set the flag to 'ok' (or 1, or whatever else you like) so next section fires
24. }
25. }
```

We have now checked all of the required fields from our form. We may have had more fields, but for whatever reason are not concerned enough about their contents to require or validate them. You may be wondering why we are validating in PHP since JavaScript can do this before the form is submitted in real time. There are two reasons (re)validating with PHP can help you. First, is the more obvious case in which the end user's browser, firewall network policy, etc. has disabled JavaScript. While your form would still function, you would end up with possibly invalid data. Second is that any bots that find your form and attempt to use it maliciously are likely to read your form's destination and send data directly to your processing script, also circumventing your validation. This allows you to add sanitization through PHP in addition to other safety precautions to further harden your site.

Next we will take a look at some options of what we can do once the form has been checked and is OK. Ultimately, there are two main tasks for you here. The first, is do whatever it is you want to do with what the user gave you—email the results, store them in a database, process a registration or login, use them to search data or affect output, etc. The second, is to provide feedback that demonstrates to the user how their action affected the system. It could be search results, a “successful” notice like “Thank You for Registering!” or the result of their interaction, like them being logged into the system.

```
1. if($check=='ok'){ // email ourselves a copy of what they submitted and tell them they are done
2. mail("us@oursite.com", "$_GET[firstName] created and account.", print_r($_GET,true), "From: noreply@oursite.com");
3. echo "Thank you for registering!";
4. }
```

Finally, our last logical test will be true if the user has not submitted anything or if there were errors. By creating this section as follows, we can support both cases at the same time:

```
1. if($errors!='ok'){ //there were errors or the form is not submitted ?>
2. foreach($errors as $error){echo "$error<br>";}
3. <form action='<?php echo $_SERVER['PHP_SELF']; ?>' method='get' name='registration'>
4. <input type='text' name='firstName' value='<?php echo $_GET['firstName']; ?>' /><br>
5. <input type='text' name='lastName' value='<?php echo $_GET['lastName']; ?>' /><br>
6. <input type='text' name='email' value='<?php echo $_GET['email']; ?>' /><br>
7. <input type='text' name='email2' value='<?php echo $_GET['email2']; ?>' /><br>
8. <input type='submit' name='Register' value='submit' />
9. </form>
10. <?php } ?>
```

In our last section, the foreach in the second line will print any errors that were added to the array. Since we have reached this point, \$errors either is an array and our entries will print to the screen, or it was never set, and will not show anything on the screen

if we are suppressing notices. If you want to avoid the notice generated when the form has not been submitted, we could wrap line 2 with an If statement:

```
1. if(!empty($errors)){foreach($errors as $error){echo "$error<br>";}}
```

In our form you will see we re-entered PHP inside of the value attribute of each input. By echoing the value of the input in our get array, if there is one, we will re-populate our form with what the user entered. Since the first section of our code already checked these values if the form was submitted, any bad entries will have already been reset to nothing, helping the user see what needs to be re-entered.

This effectively completes our one page form, validation, and response. We could add jQuery validation on top of our form elements to improve the user experience as well by validating during the form completion process, but bear in mind this is a progressive enhancement, meaning we should assume JavaScript is off, and that anything we use that works improves upon an already working system.

Drag and Drop

Certain tags in HTML5 now support the ability to be treated as drag and droppable items. Items that support the ability allow for attributes including draggable, ondragenter, ondragover, ondragstart, ondragend, and ondrop. When we want to define the actions that take place when one of these conditions is met, we need to call a JavaScript function, that we define ourselves. We will look at our example by creating it in layers, first defining the structure with HTML, then adding our CSS apply our visual, and finally we will add our JavaScript to give it full functionality.

The first piece of our structure is to define the places in our page where moveable objects are allowed to be. These will typically represent the start and end areas that we are allowed to move objects to and from, like a product page to a shopping cart icon, or just two big empty areas. We will create a simple two location page for now. To define our two areas that are drag and drop friendly, we define our divs as we are accustomed to doing and simply add the references to actions that are allowed, or that we want to instigate actions or changes in our visual cues:

```
1. <div id="startingLocation" ondragenter="return dragenter(event)" ondragover="return hover(event)" ondrop="return drop(event)"> </div>
2. <div id="endingLocation" ondragenter="return dragenter(event)" ondragover="return hover(event)" ondrop="return drop(event)"> </div>
```

Next, we will add the objects we want to interact with. They need a place to live when the page loads, so we will put them in the startingLocation div.

```
1. <div id="startingLocation" ondragenter="return dragenter(event)" ondragover="return hover(event)" ondrop="return drop(event)">
2. <div id="item1" draggable="true" ondragstart="return start(event)" ondragend="return end(event)">Item #1</div>
3. <div id="item2" draggable="true" ondragstart="return start(event)" ondragend="return end(event)">Item #2</div>
4. <div id="item3" draggable="true" ondragstart="return start(event)" ondragend="return end(event)">Item #3</div>
5. </div>
```

While this now gives us a drag and drop foundation, it is not exactly user friendly yet. If you save and test what we have, you will find a very blank screen that is probably rather difficult to interact with as we cannot tell where the different objects start and end, and even at that we have no actions. To address this, we need to add some CSS to our file:

```
1. <style type="text/css">
2. #startingLocation, #endingLocation{
3. Float:left;
4. Width:200px;
5. Height:200px;
6. Margin:10px;
7. }
8. #startingLocation{
9. Background-color:red;
10. }
11. #endingLocation{
```

```
12. Background-color:green;
13. }
14. #item1, #item2, #item3{
15. Width:60px;
16. Height:60px;
17. Padding:5px;
18. Margin:10px;
19. }
20. </style>
```

To give us functionality, we need to add JavaScript to dictate what happens when items are moved around on the screen. We need to provide the start function permission to move items, dictate what information it needs to bring with the dragged object, and what to display when the object is in motion:

```
1. <script type="text/javascript">
2. function start(event){
3. //Give the draggable object permission to move
4. event.dataTransfer.effectAllowed='move';
5. //Grabs the dragged items ID for reference
6. event.dataTransfer.setData("id",event.target.getAttribute('id'));
7. // Sets our drag image with no offset
8. event.dataTransfer.setDragImage(event.target, 0, 0);
9. return true;
10. }
11. </script>
```

Next, we need to define what happens to our objects when they are held over an area that takes drops. To do this, we will add the definition of the hover() function we referred to when we created our HTML:

```
1. function hover(){
2. //reads the ID we provided of the dragged item
3. var idraggable = event.dataTransfer.getData("id");
4. // reads the ID of the object we are hovering over
5. var id = event.target.getAttribute('id');
6. //All items can be dropped into endingLocation
7. if(id=='endingLocation') return false; else return true;
8. }
```

If we wanted to declare that only our first two draggable items are allowed into the endingLocation box, we would change our if statement to specify which items are allowed:

```
1. If(id=='endingLocation')&& (idraggable=='item1' || idraggable=='item2') return false;
```

Next we need to complete the act of moving our item to its new location. We will add one more function we have already made reference to in our HTML, drop():

```
1. function drop(){
2. var idraggable event.dataTransfer.getData('id');
3. event.target.appendChild(document.getElementById(idraggable));
4. event.stopPropagation();
5. return false;
6. }
```

Finally, we need to clean up. Now that our item is dropped, we do not need to worry about its value any longer:

```
1. function end(){
2. event.dataTransfer.clearData('id');
3. return true;
4. }
```

If we were going to use our drag and drop system as a shopping cart, we would want to flesh out more actions in our end function. We would add code to add the item to the session or cookie record of our shopping cart, and could trigger other actions like updating our cart total on the screen or prompting for a number of that item we want in our cart.

5.2: [Integration Examples](#) is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- 5.2: [Integration Examples](#) by [Michael Mendez](#) has no license indicated.