Building a Fully Functioning Website:

Static VS. Dynamic

Up to now, we've worked with static pages built with HTML, CSS, Bootstrap, and Javascript. The sites look great, but the downside is that the sites have no functionality beyond being responsive on the front end. The sites we've built so far are what are known as **static** sites. That is, while the sites may be responsive to the user, the data on the site will always remain the same since there's no way to change it on the server that is serving up the site.

Dynamic sites on the other hand, are sites that serve up not only pages for the user to view, but also the sites allow a level of interactivity with the site. This interactivity requires not only sending data to the user, but then receiving data from that user, saving it, and working with it in some way. One of the most common functions you'll use on developing a site is the use of a login page so that users can access specific material!

PHP and Websites

In order to use PHP, we not only need to make sure our files have a `.php` file extension, but also are wrapped in a `<?php` opening tag, and a `?>` closing tag. This tells the interpreter where to look for the php code to execute:

```
<!doctype html>
<html>
  <body>
    <?php
      echo "<p> hello world </p>";
      ?>
  </body>
</html>
```

PHP can not only execute code within an HTML encoded document, it can also render HTML so long as its properly printed. Notice the "Hello World" within the string. When called, that string will render to HTML and the site will look just as if it were:

```
<!doctype html>
<html>
  <body>
    <p> hello world </p>
  </body>
</html>
```

All of this rendering gets done server side, and once complete, it is sent to the user as if it were any other html document!

Connecting to the database:

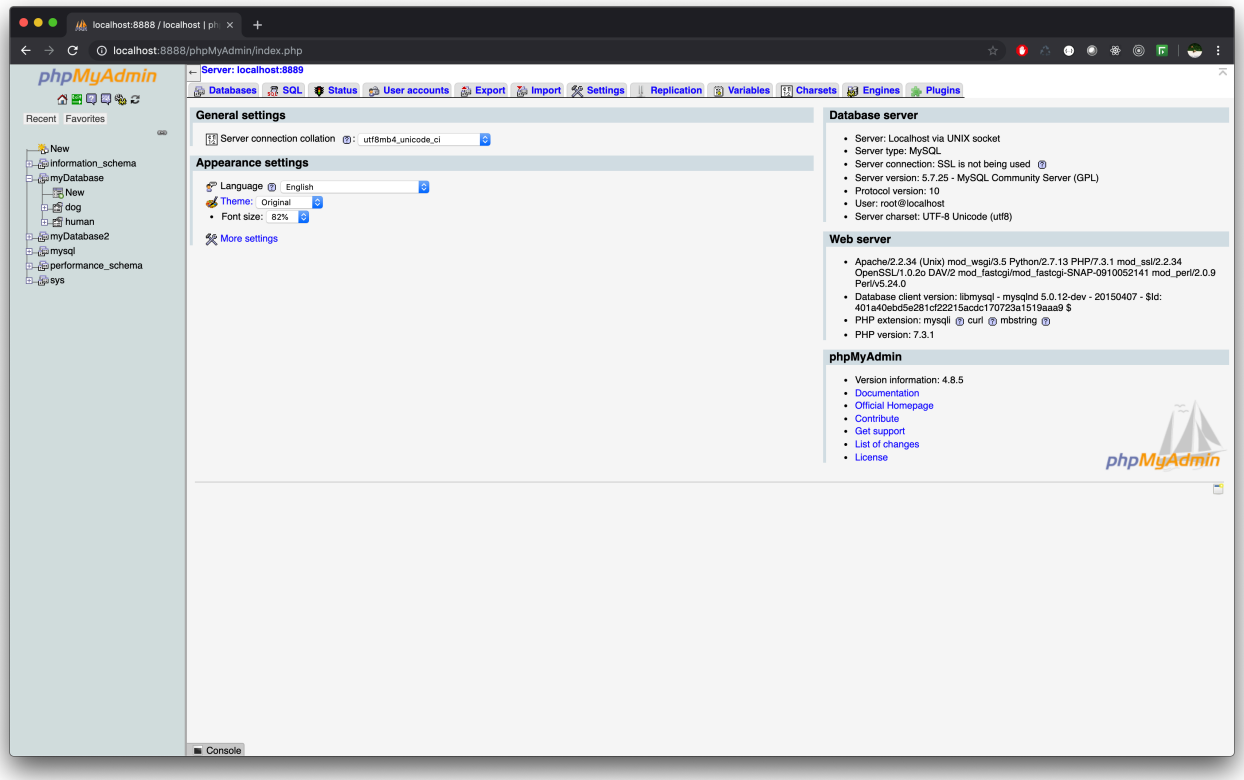
In order to connect our PHP to our database, we'll need to do a few things:

1. Make sure the XAMPP/MAMP server is running (so that we can execute our php and access our database).
2. Create a user login for our database.

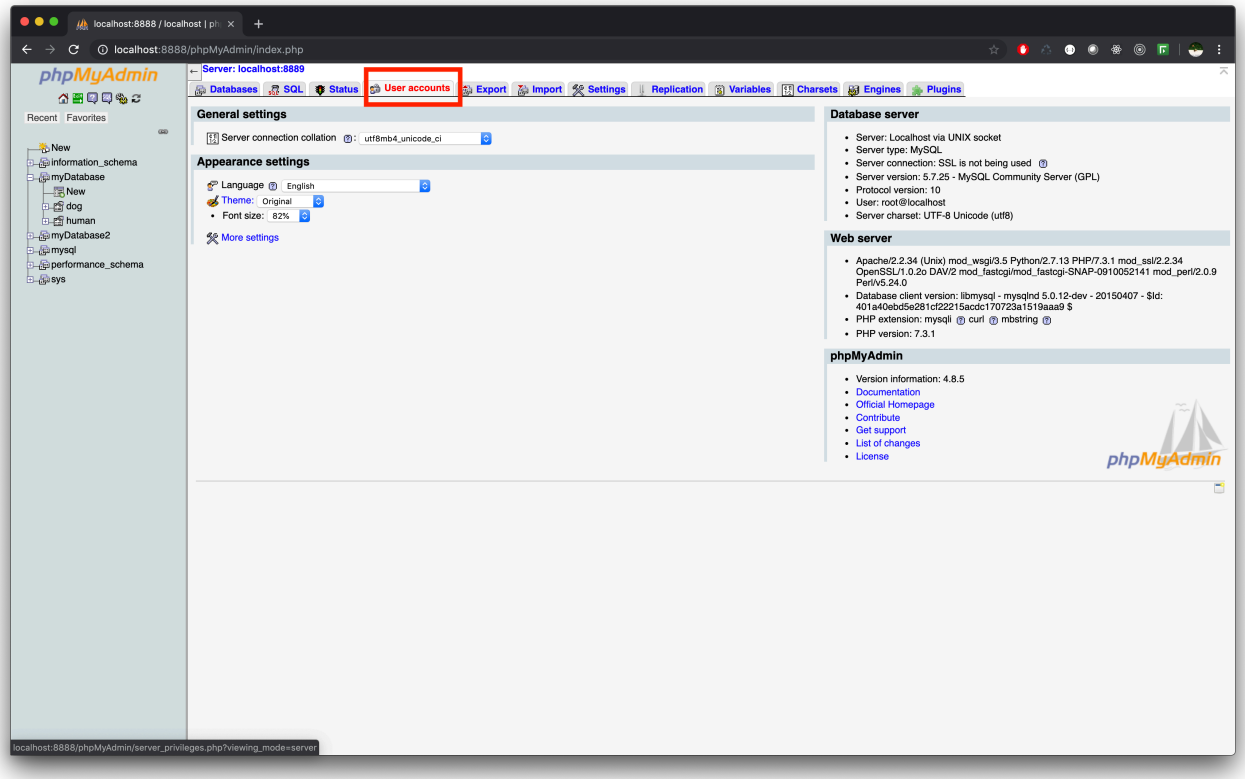
The first step is easy enough, but the second step, however, requires some digging.

Creating a Database User:

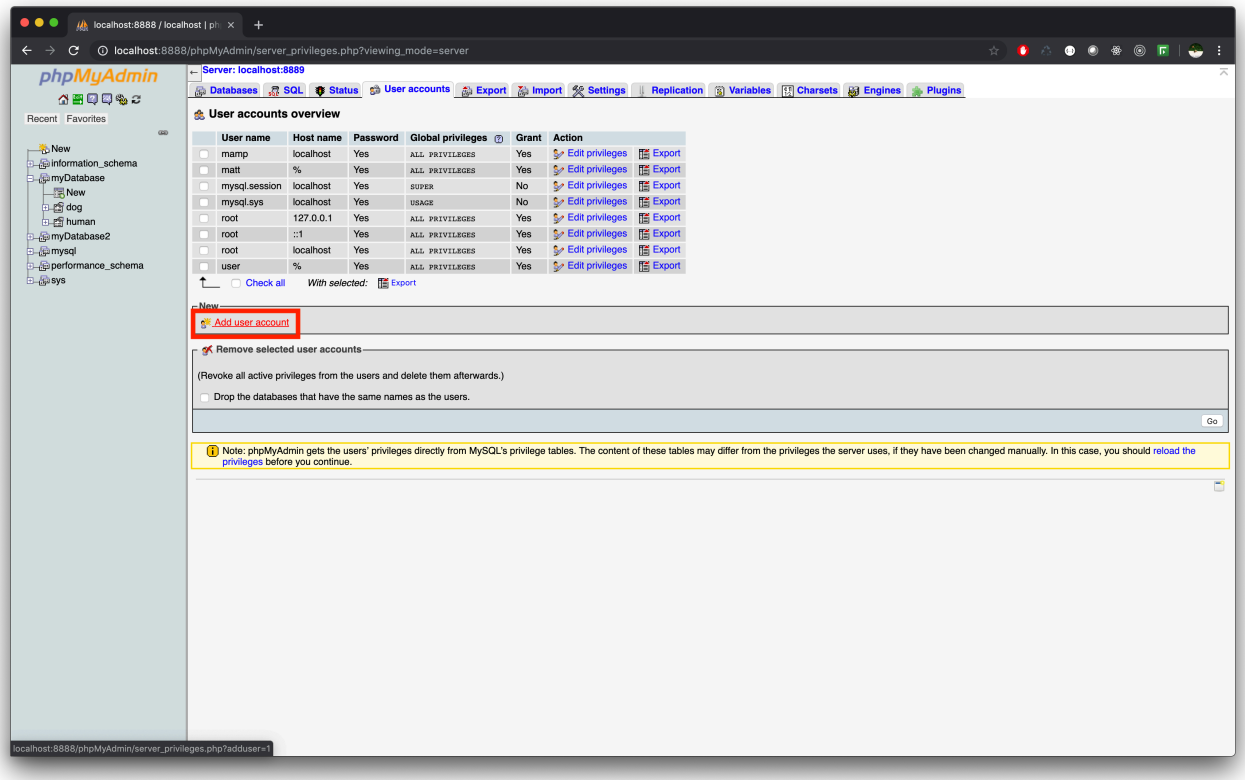
Go to phpMyAdmin home page (that is, make sure you're not in a single database's page). To make sure you're on the main page, click the phpMyAdmin logo.



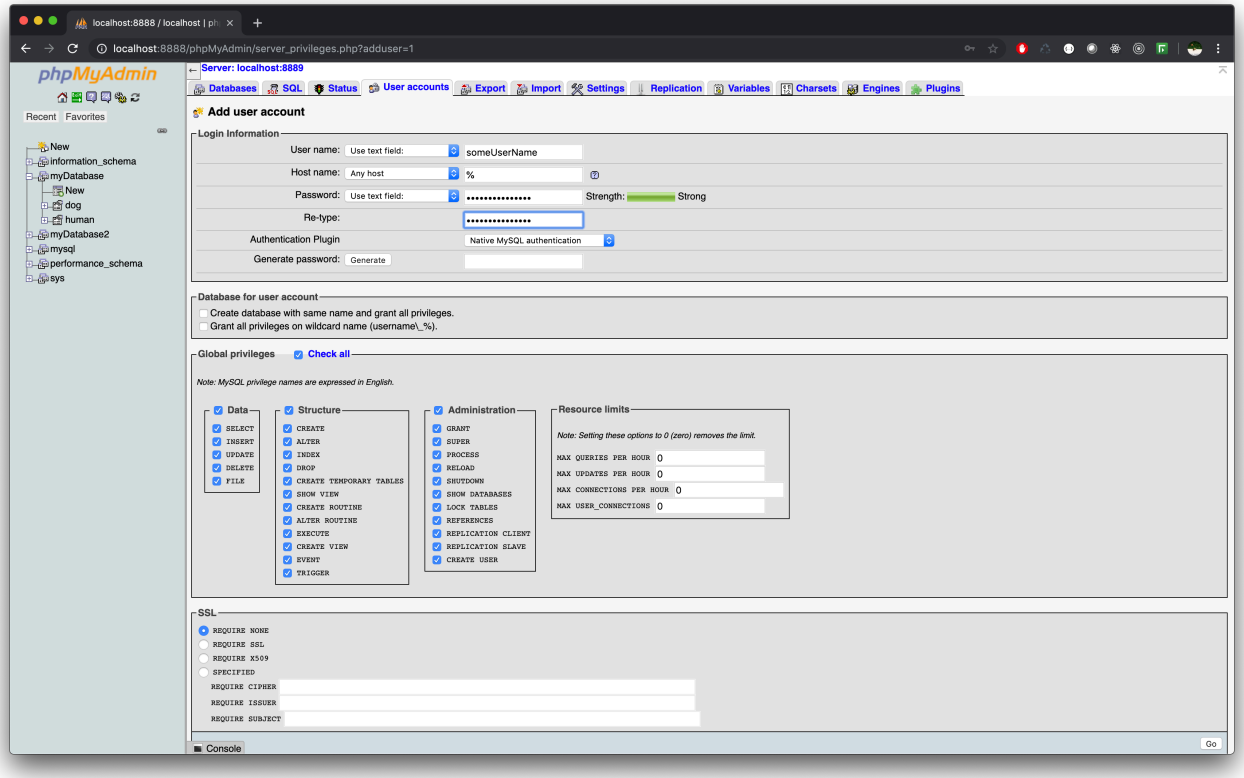
Then click into the user accounts section:



Once in the user accounts section, navigate to the `add user account` link:



Once there, fill out the page's instructions. Make sure to select the `Check all` on `Global Privileges`:



Once finished you ought to have the ability to log into your MySql Database.

Using PHP to connect:

We can connect to our database very easily with phpMyAdmin. Viewing and interacting with our database via a GUI is great, but we want our website to do that programatically. Let's give it a try. In the `htdocs` folder, create a file called `databaseConnector.php`. In this file, we're going to connect to our database. We'll use the included extension `mysqli`:

```
<!doctype html>
<html>
  <body>
    <?php
      $servername = 'localhost';
      $username = 'myUsername';
      $password = 'myPassword';
      $myDatabase = 'myDatabase';

      // $conn = new mysqli('localhost', 'root', '');
      $conn = new mysqli($servername, $username, $password, $myDatabase);

      if ($conn->connect_error) {
          die('Connections failed: '.$conn -> connect_error);
      }
      echo "Connection successful. You're a real champ. <br />";

      $conn->close();
  ?>
  </body>
</html>
```

What we've done in the above code is:

- Create variables for our server name, username, password, and specific database
- Create a new `mysqli` object to connect to the database, and store that connected object in a variable
- Check whether we've connected or not by checking whether a connection error exists
- Close the connection.

In the above code, we've opened a link to our database, checked our connection, and then closed our link. It is very important to make sure you close your connections to your database, otherwise, you may have too many open at one point and wind up blocking any new connections from coming in. Note: If your page printed "Connection Failed" followed by an error message, make sure to read the message. It's possible you typed your username/password/database/etc incorrectly.

Querying the Database with code:

Select:

Connecting to a database is great, but without submitting a query to the database, we've really only proved that we can connect to it. In order to query the database, you'll need a sql query to begin with. For now, we'll just use (from our dog table created in the mysql lecture):

```
1 SELECT * FROM dog WHERE 1;
```

In order to send that mysql query to our page, we'll need to first save it to a variable, and then send it over with a supplied function from `$conn`:

```
1  <!doctype html>
2  <html>
3    <body>
4      <?php
5        $servername = 'localhost';
6        $username = 'myUsername';
7        $password = 'myPassword';
8        $myDatabase = 'myDatabase';
9
10       $conn = new mysqli($servername, $username, $password, $myDatabase);
11
12       if ($conn->connect_error) {
13           die('Connections failed: '.$conn -> connect_error);
14       }
15       echo "Connection successful. You're a real champ. <br />";
16
17       $sqlStatement = "SELECT * FROM dog WHERE 1";
18       echo "Querying database with: $sqlStatement";
19
20       $result = $conn->query($sqlStatement);
21
22
23       $conn->close();
24   ?>
25   </body>
26 </html>
```

Note that we're making the query at line 21, and storing the response in `$result`. If you find yourself trying to echo result, it's very likely you won't see anything at all. That's because `$result` is an object, and our data aren't actually at the top level. To work with the data, you'll need to make sure that data exist in the firs place, and if they do, then iterate over each returned row:

```
 <!doctype html>
<html>
  <body>
    <?php
      $servername = 'localhost';
      $username = 'myUsername';
      $password = 'myPassword';
      $myDatabase = 'myDatabase';

      $conn = new mysqli($servername, $username, $password, $myDatabase);

      if ($conn->connect_error) {
          die('Connections failed: '.$conn -> connect_error);
      }
      echo "Connection successful. You're a real champ. <br />";

      $sqlStatement = "SELECT * FROM dog WHERE 1";
      echo "Querying database with: $sqlStatement";

      $result = $conn->query($sqlStatement);

      $conn->close();

      if ( $result -> num_rows > 0 ) {
        while($row = $result->fetch_assoc() ) {
          $dogName = $row["name"];
          $age = $row['age'];
          $breed = $row['breed'];
          $humanID = $row["humanID"];

          echo "<div>".$dogName.$age.$breed.$humanID."</div> ";
        }
      }
    ?>
  </body>
</html>
```

There are three major things of note in the above code, and one minor:

1. In order to make sure you've received data from the database, it is always good to preface your logic with a conditional to ensure that data were received back.
2. The while loop at 26 is running off of the condition that something is being stored in `$row`. Every time the loop's condition is called, a new row is saved into `$row` by the `fetch_assoc()` method. This pulls a single row at a time, so that we can manipulated its individual data.
3. All database return objects for each row act as dictionaries with the column names as the keys

for the values.
4. Minor note: See that we closed our database connection on line 23 before any of our logic began? That's to ensure that we're only leaving our connection open for the duration of the call to get the response to not take up valuable real estate .

Now, upon calling this page's code, we can see a list of dogs, their ages, and their breeds pop up. This is all coming from the database, programmatically rendered by the php into a series of consecutive divs, and then served back to the user.

If you don't see a list of dogs being rendered, make sure to check that your SQL statement is the correct syntax.

Cleaning Up Code

Copying and pasting the same code over and over again can become significantly repetitive and make bug hunting very difficult. To get around this, it's good to break code up into smaller, reusable functions! If we take a look at the above code, it's possible to break it up into functions for greater reusability! Let's create a file called `databaseQuery.php` to put our functions in. We can break our code up into, for now, three reusable functions: Connection, Query, Disconnection:

```
1  <?php
2      function connect() {
3          $servername = 'localhost';
4          $username = 'user';
5          $password = 'password';
6          $mydatabase = 'myDatabase';
7
8          $conn = new mysqli($servername, $username, $password, $mydatabase);
9
10         return $conn;
11     }
12
13     function disconnect($conn) {
14         $conn->close();
15     }
16
17     function dbSelect($what, $condition, $table){
18         $conn = connect();
19
20         if ($conn -> connect_error) {
21           die('connection failed: '.$conn->connect_error);
22         }
23         $sqlStatement = "SELECT $what FROM $table WHERE $condition;";
24         echo "Querying database with: $sqlStatement";
25         $result = $conn->query($sqlStatement);
```

```
26
27          disconnect($conn);
28
29          return $result;
30      }
31  ?>
```

Now, when we call this from a separate page, we can connect, disconnect, and query our database by calling only one function, `dbSelect()` with parameters as opposed to having to copy paste the same code over and over again.

Let's take the remainder of our code above and create a function that will take the result of the above code and render it into a table and store it in a file called `dogData.php`:

```
1   <?php
2       function getDogDataTable($result){
3        echo "<table>";
4                   $total = 0;
5
6                   if ( $result -> num_rows > 0 ) {
7                   while($row = $result->fetch_assoc() ) {
8                       $dogName = $row["name"];
9                       $age = $row['age'];
10                      $breed = $row['breed'];
11                      $humanID = $row["humanID"];
12                      // $br = "<br />";
13                      $total += $age;
14                      echo "<tr> <td> $dogName</td>  <td> $age</td>
    <td>$breed</td>  <td>$humanID </td>  </tr>";
15                  }
16                  echo "</table>";
17
18                  $aver_AGE = $total / $result->num_rows;
19                  echo "Average dog age: $aver_AGE ";
20          }
21      }
22  ?>
```

Finally, let's create a page called `dogInformation.php`. In this page, we'll call our database queries and render the return values:

```php
<!DOCTYPE html>
<html>
  <body>
    <?php
      include "databaseQuery.php";
      include "dogData.php";
      $dogData = dbSelect("*", 1, "dog");
      getDogDataTable($dogData);
    ?>
  </body>
</html>
```

Now when we go to `localhost/dogInformation.php` (you might have to call `localhost:8888/dogInformation.php` or some other port depending on how you had it set up. XAMPP and MAMP display which ports they're using), the code will call dbSelect, which in turn connects to the database, retrieves the data, and disconnects all within the `dbSelect()` call, and then we render the dog table with the `getDogDataTable()` function call. By creating functions return php+html, we're able to code much easier to read/code websites.

From here on out, we'll be using functions for our html renderings to keep our pages clean looking.

Insert

Selecting data from the database is useful, but so is inserting data (how else would the data get to the database in the first place). Let's create a place on our `dogInformation.php` to enter in new dog information:

```php
<!DOCTYPE html>
<html>
  <body>
    <?php
      include "databaseQuery.php";
      include "dogData.php";
      $dogData = dbSelect("*", 1, "dog");
      getDogDataTable($dogData);
    ?>

    <div> Oh hey - what's your dog's information?</div>

    <form action="uploadDog.php" method="get">
      <input type='text' value='name' name='name'/>
      <input type='text' value='age' name='age'/>
```

```
      <input type='text' value='breed' name='breed'/>
      <input type='text' value='humanID' name='humanID'/>
      <input type = 'submit'>
    </form>

  </body>
</html>
```

On this page, we've not got an area for our user to enter information, and then submit it through a form. This is a perfect time to review forms.

Forms, revisited:

A form is an element in html that can execute actions of some time with specific methods in mind. Notice that our form's action is `uploadDog.php`. That action is going to send us to the page `uploadDog.php` where some code is then expected to be executed. For now we don't have an uploadDog.php page , but soon enough we will!

The `method` attribute takes in a particular http request method type and sends the data in such a way that is congruent with that type. There are number of http request methods, but the ones that we will work with are:

- GET: Sends data via the URL request.
- POST: Sends data via a request "Body" that stores the information. Typically used create / update or possibly for larger requests.
- PUT: Sends data via the request body. PUT is typically used to update or possibly create a resource.
- DELETE: Deletes a resource.

The very bottom of the form, there is an input type called submit. When clicking this button, the data from the form get send to whatever action is being called (in this case, `uploadDog.php`).

Notice that when we submit our action from the above code, we get a URL that says:

```
http://localhost:8888/uploadDog.php?
name=Airbud&age=4&breed=Golden+Retriever&humanID=12
```

Now that we're sending data, let's create a page to work with that data. To access material sent via http request types, we need to use special php keywords. These keywords are dictionaries that have all of the data inside of them. Take a look at the new file `uploadDog.php`:

```php
<?php
    $dogName = $_GET['name'];
    $dogAge = $_GET['age'];
    $dogBreed = $_GET['breed'];
    $humanID = $_GET['humanID'];

    echo "We were passed: $dogName, $dogAge, $dogBreed, $humanID <br />";
?>
```

To access the data from the get request, we call `$_GET`. The same goes for other HTTP requests too (we'll see a $_POST in action quite soon).

We're getting dog data out of the `$_GET` dictionary, let's do something with it. First, let's import our `databaseQuery.php` file:

```php
<?php
    include 'databaseQuery.php';

    $dogName = $_GET['name'];
    $dogAge = $_GET['age'];
    $dogBreed = $_GET['breed'];
    $humanID = $_GET['humanID'];

    echo "We were passed: $dogName, $dogAge, $dogBreed, $humanID <br />";
?>
```

Now that we've imported `databaseQuery`, let's go back to that file, and create an `insert` method:

```php
<?php
    function connect() {
        $servername = 'localhost';
        $username = 'user';
        $password = 'password';
        $mydatabase = 'myDatabase';

        $conn = new mysqli($servername, $username, $password, $mydatabase);

        if ($conn -> connect_error) {
          die('connection failed: '.$conn->connect_error);
        }

        return $conn;
    }

    function disconnect($conn) {
```

```
        $conn->close();
    }

    function dbSelect($what, $condition, $table){
        $conn = connect();

        $sqlStatement = "SELECT $what FROM $table WHERE $condition;";
        echo "Querying database with: $sqlStatement";
        $result = $conn->query($sqlStatement);

        disconnect($conn);

        return $result;
    }

    function dbInsert($whatArray, $table) {
      $conn = connect();

      $sqlStatement = "INSERT INTO $table (`name`, `breed`, `age`, `humanID`)
VALUES ($whatArray[0],$whatArray[1],$whatArray[2],$whatArray[3])";
      echo "Querying database with: $sqlStatement";
      $result = $conn->query($sqlStatement);

      disconnect($conn);

      return $result;
    }

?>
```

With a new dbInsert function we can use that in our `uploadDog.php` file:

```php
<?php
    include 'databaseQuery.php';

    $dogName = $_GET['name'];
    $dogAge = $_GET['age'];
    $dogBreed = $_GET['breed'];
    $humanID = $_GET['humanID'];

    echo "We were passed: $dogName, $dogAge, $dogBreed, $humanID <br />";

    $callArray = [$dogName, $dogBreed, $dogAge, $humanID];
    dbInsert($callArray, "dog");
?>
```

Taking time to refactor:

You might have noticed that our two functions in the `databaseQuery.php` look surprisingly similar. When you notice code like this, it's always good to refactor. Both dbInsert and dbSelect have very similar code except for their sql queries. A way for us to deal with that would be to update our functions into one:

```php
<?php
    function connect() {
        $servername = 'localhost';
        $username = 'user';
        $password = 'password';
        $mydatabase = 'myDatabase';

        $conn = new mysqli($servername, $username, $password, $mydatabase);

        if ($conn -> connect_error) {
          die('connection failed: '.$conn->connect_error);
        }

        return $conn;
    }

    function disconnect($conn) {
        $conn->close();
    }

    function dbQuery($sqlQuery){
        $conn = connect();
```

```
23
24          $result = $conn->query($sqlQuery);
25
26          disconnect($conn);
27
28          return $result;
29      }
30  ?>
```

We've updated our function to no longer take in an array or a specified list of parameters. Now all we need to do is pass in a sql query. Because we've changed the function header, however, we'll need to change every file that touches this one (uploadDog and dogInformation):

```
1   <?php
2       include 'databaseQuery.php';
3
4       $dogName = $_GET['name'];
5       $dogAge = $_GET['age'];
6       $dogBreed = $_GET['breed'];
7       $humanID = $_GET['humanID'];
8
9       echo "We were passed: $dogName, $dogAge, $dogBreed, $humanID <br />";
10
11      $sqlQuery = "INSERT into dog (name, breed, age, humanID) VALUES
    ('$dogName', '$dogBreed', '$dogAge', '$humanID')";
12
13      dbQuery($sqlQuery);
14  ?>
```

AND

```
1   <!DOCTYPE html>
2   <html>
3     <body>
4       <?php
5         include "databaseQuery.php";
6         include "dogData.php";
7         $sqlQuery = "SELECT * FROM dog WHERE 1";
8         $dogData = dbQuery($sqlQuery);
9         getDogDataTable($dogData);
10      ?>
11
12      <div> Oh hey - what's your dog's information?</div>
13
14      <form action="uploadDog.php" method="get">
```

```
      <input type='text' value='name' name='name'/>
      <input type='text' value='age' name='age'/>
      <input type='text' value='breed' name='breed'/>
      <input type='text' value='humanID' name='humanID'/>
      <input type = 'submit'>
    </form>

  </body>
</html>
```

While it may not look as clean as it did earlier when calling dbQuery, we can now use our `sqlQuery` function for anything we want, and not just for specific inserts/updates/etc.

Session Variables

Database queries offer a lot of power to a site, but what if you wanted a variable to remain persistent throughout the whole site, without having to make constant database calls? We can do that with session variables. These variables allow for us to store data across several pages throughout the user's session.

Before we start working with session variables, let's take a moment and head back to our database and create a new table called `login` and add to it the columns `username` and `password`.

```
CREATE TABLE `myDatabase`.`login` ( `username` TEXT NOT NULL , `password` INT NOT NULL ) ENGINE = InnoDB;
```

To demonstrate these variables, let's first take a moment and add a navbar to our entire site. Let's create a file called `navbar.php`:

```
<nav class='navbar navbar-default'>
  <div class='navbar-header' style='background-color:#f1f1f1'>
    <a href='index.php' class='navbar-brand'>Site</a>
  </div>

  <ul class='nav navbar-nav'>
    <li> <a href='index.php'> Photo Gallery </a></li>
    <li> <a href='dogInformation.php'> Dogs </a></li>
  </ul>

  <ul class='nav navbar-nav navbar-right'>
    <li> <a href='login.php'> Log In </a></li>
  </ul>
```

```
</nav>
```

Two things to notice here. First we're using bootstrap classes. When we use this file in any other page, we need to be sure we're pulling in the required bootstrap styling. Secondly, we don't have any php at the moment, but we will soon enough. Up to now, we don't have an index page. Let's create that, and call it `index.php` by taking our index html from the bootstrap lesson and changing it up a little bit:

```
<!doctype html>
<html>
  <head>
    <title>Photo Blog!</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.cs
s" integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">

    <script  src="http://code.jquery.com/jquery-3.3.1.js"
integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
crossorigin="anonymous"></script>


    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"
integrity="sha384-
aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd"
crossorigin="anonymous"></script>

    <style>
      img { width: 500px; height: 500px }
      .border{ border: 2px solid black; margin: 5px}
    </style>

  </head>
  <body>

    <?php include './navbar.php' ?>

    <div class='container'>
      <div class='jumbotron'>
        <h1>
          My Photo Blog:
```

```
        </h1>
        <h4>
          This is a place for me to show off my photos.
        </h4>
      </div>
      <hr />
      <div class='container'>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/batwing.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/christmasTarantula.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/cicada.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/desertHairy.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/giantFlower.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
```

```
          <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/glassWing.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
          <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/greenJay.png' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
          <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/leafwing.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
          <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/lubber.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
          <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/luna.JPG' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/morphoScales.png' />
          </div>
        </div>
        <div class='col col-lg-4 col-sm-6'>
          <div class='thumbnail'>
            <img
src='https://raw.githubusercontent.com/LaneMatthewJ/lanematthewj.github.io/master/img/moth.JPG' /></div>
        </div>
```

```
      </div>
    </div>
  </body>
</html>
```

Now when we open our index.php page, we're pulling in all of the navbar code! By writing code like this, we can write a web page without having to copy paste the navbar code for every single one.

Now let's create a `login.php` page for our navbar to link to with an HTML form to call a log in function:

```
<!DOCTYPE html>
<html>
    <head>
        <title> Login Page </title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.cs
s" integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">

    <script  src="http://code.jquery.com/jquery-3.3.1.js"
integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
crossorigin="anonymous"></script>


    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"
integrity="sha384-
aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd"
crossorigin="anonymous"></script>

    </head>
    <body>
        <?php
            include "navbar.php";
        ?>

        <div>FILL THIS OUT! </div>
        <form action="loginFunction.php" method="post">
            USERNAME <input type='text' name='name' value="ENTER USERNAME
HERE"/> <br />
            PASSWORD <input type='password' name='password' value="ENTER PW
HERE"/>
            <input type="submit">
```

```
        </form>
    </body>
</html>
```

Our form here is using a post, so when we create our `loginFunction.php` page, we'll have to access our data with `$_POST` as opposed to earlier when we used `$_GET`. Let's create our login page:

```
<?php
    include "databaseQuery.php";

    $siteUser = $_POST["name"];
    $userPassword = $_POST["password"];

    $sqlQuery = 'SELECT username FROM login WHERE username="'.$siteUser.'" AND password="'.$userPassword.'"';

    $dbResults = dbQuery($sqlQuery);

    if ($dbResults->num_rows == 0 ) {
        return -1;
    }

    $row = $dbResults->fetch_assoc();
    $username = $row["username"];

    echo "<h1> Welcome ".$username." </h1><br />";

    session_start();
    $_SESSION["username"] = $username;
?>
```

Notice at line 20, we've used the function `session_start()`. This function starts a session and whatever we save into the `$_SESSION` dictionary (i.e. 'username'), we'll then be able to retrieve at any other place in our site!

Because we don't want our users to sit on the login page (it's not very good looking now), let's add a redirect back to our index.php (also, let's remove the echo since we're redirecting):

```
<?php
    include "databaseQuery.php";

    $siteUser = $_POST["name"];
    $userPassword = $_POST["password"];

```

```
    $sqlQuery = 'SELECT username FROM login WHERE username="'.$siteUser.'"
AND password="'.$userPassword.'"';

    $dbResults = dbQuery($sqlQuery);

    if ($dbResults->num_rows == 0 ) {
        return -1;
    }

    $row = $dbResults->fetch_assoc();
    $username = $row["username"];

    session_start();
    $_SESSION["username"] = $username;

    header("Location: ./index.php");
    exit();
?>
```

By placing the `exit()`; after the `header("Location: ./index.php");` we absolutely ensure that nothing after our code will accidentally get executed.

Let's move back to our navbar and actually change things up a bit so we can actually render our username!

```
    <nav class='navbar navbar-default'>
      <div class='navbar-header' style='background-color:#f1f1f1'>
        <a href='index.php' class='navbar-brand'>Site</a>
      </div>

      <ul class='nav navbar-nav'>
        <li> <a href='index.php'> Photo Gallery </a></li>
        <li> <a href='dogInformation.php'> Dogs </a></li>
      </ul>

      <ul class='nav navbar-nav navbar-right'>
      <?php
      session_start();
      $username = $_SESSION['username'];

        if (strlen($username) >0 ){
            echo   "<li><a href='#'> ".$username."</a></li>";
            echo  "<li> <a href='./logout.php'> Log Out </a> </li>";
        } else {
```

```
            echo " <li> <a href='./login.php'> Log In </a></li>'";
        }

      ?>
      </ul>
    </nav>


```

What we did above is add php to our navbar so that when a user is logged in, we'll be able to properly see their username instead of the "Log In" link on the tab! Notice too that we added a `logout` link on line 18. Users can't stay logged in forever, so let's create that file now:

```
<!DOCTYPE html>
<html>
    <head>
        <title> Logout Page </title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.cs
s" integrity="sha384-
HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">
    <script  src="http://code.jquery.com/jquery-3.3.1.js"
integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
crossorigin="anonymous"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"
integrity="sha384-
aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd"
crossorigin="anonymous"></script>
    </head>

    <body>
        <?php
            include "navbar.php";
            $username = $_SESSION['username'];
            echo "<h1> Sorry to see ya go! ".$username. " </h1>";

            session_destroy();
        ?>
    </body>
</html>
```

The `session_destroy();` function removes all data stored in the `$_SESSION`.

Rendering based on User Credentials:

So far, we've been creating components and pulling them into our pages one by one. With this and a little bit extra php, we can create pages that show completely different views depending on who the user is. First, let's go back to our database's login table and add a new column called 'admin':

```
ALTER TABLE `login` ADD `admin` INT NOT NULL AFTER `password`;
```

Now, let's alter our `login.php` page to query for admin as well:

```
<?php
    include "databaseQuery.php";

    $siteUser = $_POST["name"];
    $userPassword = $_POST["password"];

    $siteUser = $_POST["name"];
    $userPassword = $_POST["password"];

    $sqlQuery = 'SELECT username, `admin` FROM login WHERE
username="'.$siteUser.'" AND password="'.$userPassword.'"';

    $dbResults = dbQuery($sqlQuery);
    if ($dbResults->num_rows == 0 ) {
        return -1;
    }

    $row = $dbResults->fetch_assoc();
    $username = $row["username"];
    $isAdmin = $row['admin'];
    echo "Welcome ".$username."<br />";
    echo "Admin? ".$admin;

    session_start();
    $_SESSION["username"] = $username;
    $_SESSION['isAdmin'] = $isAdmin;


    header("Location: ./index.php");
    exit();

```

```
?>
```

So, now, if we have an admin user, we can get that data out of the session variable `$_SESSION['isAdmin']`. One thing to note here is that we've wrapped `admin` in ticks. We want to do that because `admin` is a reserved word. To make sure we don't upset mysql, we need to make sure that the database knows we're asking for the column, and not a reserved word.

With our new found administrator rights, let's go back to our `dogInformation.php` page and hide the upload dog form behind administrator rights! First, though, let's break the page up into two separate pages, one called `dogInformation.php` which displays dog information (like ours already does), and a separate page called `uploadDogForm.php`:

```
<?php
    session_start();
    $username = $_SESSION['username'];
    echo
   " <div> Oh hey $username what's your dog's information?</div>

    <form action='uploadDog.php' method='get'>
      <input type='text' value='name' name='name'/>
      <input type='text' value='age' name='age'/>
      <input type='text' value='breed' name='breed'/>
      <input type='text' value='humanID' name='humanID'/>
      <input type = 'submit'>
    </form> "
?>
```

The above form will render from the below code if and only if the user during the session is an admin user!

```
<!DOCTYPE html>
<html>
  <body>
    <?php
      include "databaseQuery.php";
      include "dogData.php";
      $sqlQuery = "SELECT * FROM `dog` WHERE 1";
      echo "SQL QUERY: $sqlQuery";
      $dogData = dbQuery($sqlQuery);
      echo "SQL QUERY: $sqlQuery";

      getDogDataTable($dogData);
      echo "SQL QUERY: $sqlQuery";
```

```
14
15        session_start();
16        $isAdmin = $_SESSION['isAdmin'];
17
18        if($isAdmin){
19          include './uploadDogForm.php';
20        }
21      ?>
22    </body>
23  </html>
```