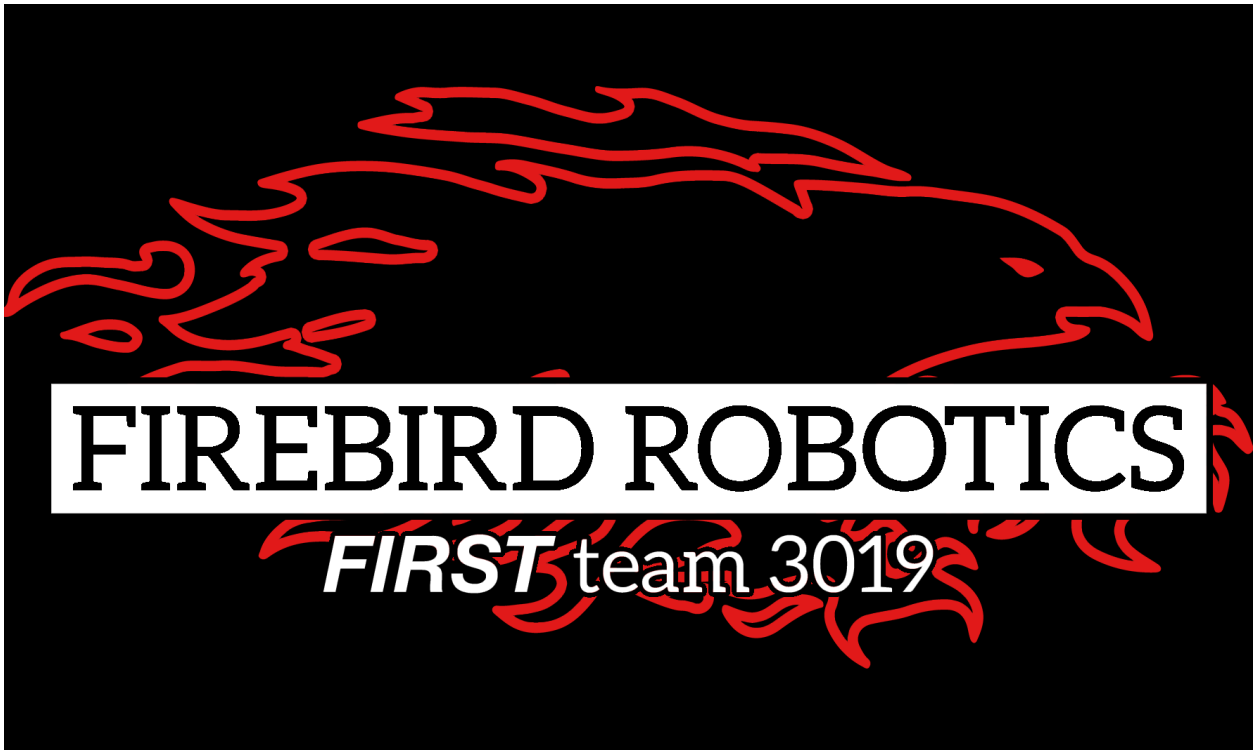
Firebird Robotics Scouting App Guide

(for less-experienced and/or newer programmers)



Branden Yang

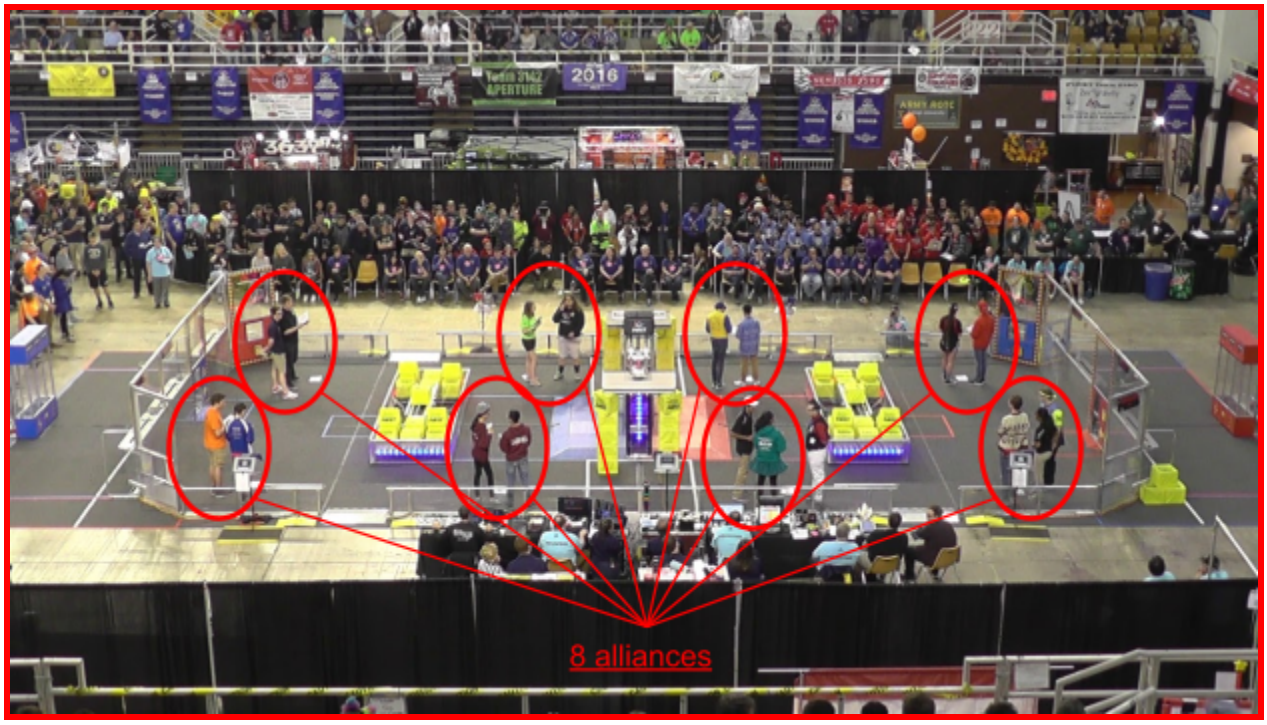
Table of Contents

***** = The modification of the scouting app should occur on a yearly basis, and as such, this section is particularly important.

What is Scouting?

In the FIRST® Robotics Competition, also known as FRC, teams compete in regional events to qualify for 1 of the 2 world championship events, depending on their location (one in Houston and one in Detroit).

During these regional events, teams play a number of qualifying matches and receive ranks based on their performance (how their performance is rated varies based on the game, which changes from year to year).



^ This is an image of an alliance selection at a regional competition from the 2018 game, FIRST Power Up. You can see the 8 different alliances circled in red, with 1 person representing each team.

After the qualifying matches have been completed, the top-rated 8 teams, referred to as alliance captains, create alliances made up of 3 different teams to participate in a final tournament, and the winning alliance is selected to go to the world championship.

The “scouting app” that will be discussed in-depth in this guide is a tool meant to assist the assessment of other teams for alliance selection. However, this doesn’t mean that if we don’t place in the top 8 teams then all of the data that was collected is useless; in many (most) cases, teams in the top 8 will be selected by other teams that also placed in the top 8, and when this occurs, the next highest rated team will take the place of the alliance captain that was chosen by another alliance. Often times, this allows teams that placed beyond 8th place to become an alliance captain (plus, it doesn’t hurt to know more about other teams’ robots).

General Information About the Scouting App

Our scouting "app" is a web-based application coded in the programming languages of HTML, CSS, JavaScript, PHP, and SQL (you can find resources for learning these languages on page 9). It is not an "app" referring to the apps published on the App Store or Google Play Store, although an iOS version of the app does exist*.

The app itself was created from scratch, but makes use of various open-source code for designing web pages and other similar uses that can be found online.

Since it is posted on a public GitHub repository, the code for the scouting app is technically open-source. If you would like to see the actual code, you may visit this link:
github.com/FirebirdRobotics/Scouting_App

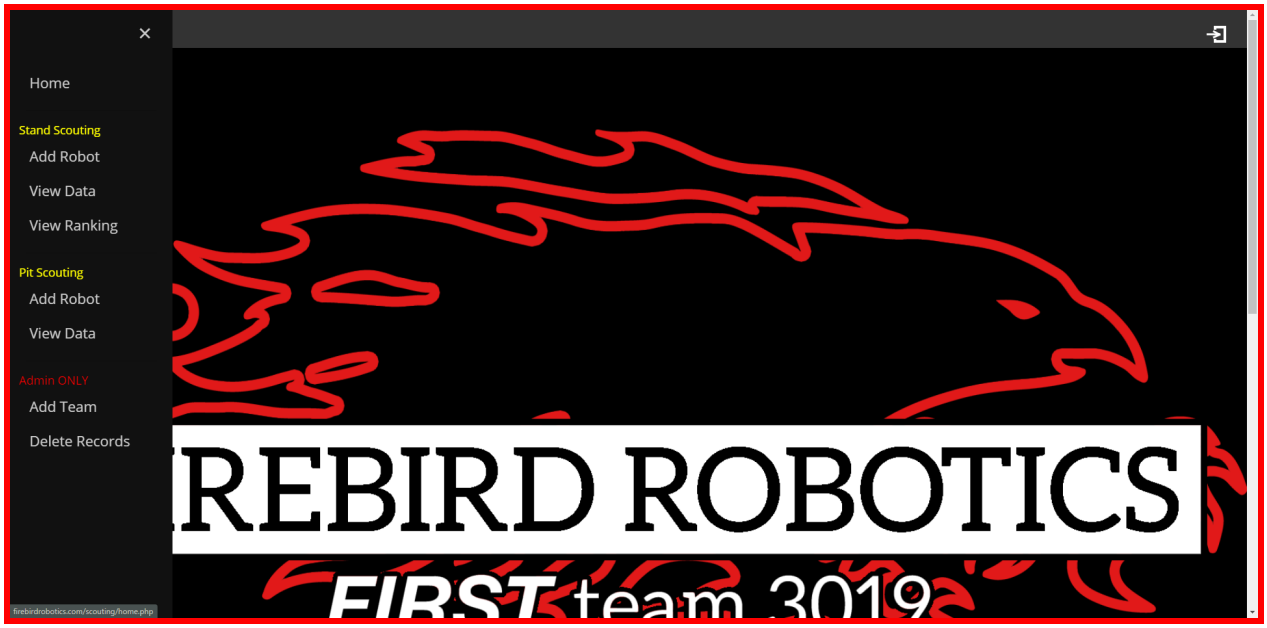
To use the scouting app, open up a browser and head to firebirdrobotics.com/scouting. From there, you'll need to create an account if you don't already have one, and then just sign in.

Note: When signing up, you should receive the "Signup Code" from one of your club officers or whoever is in charge of scouting (as the code is subject to change)

* = The iOS version of the app only functions as a reference to the webpage, so PLEASE DON'T CHANGE THE URL of the scouting app. Also, it is published under my personal account, so it is subject to deletion.

How to Use the Scouting App

After signing in to the app, you will arrive at a home page with a description of scouting at the bottom, below our team's logo.



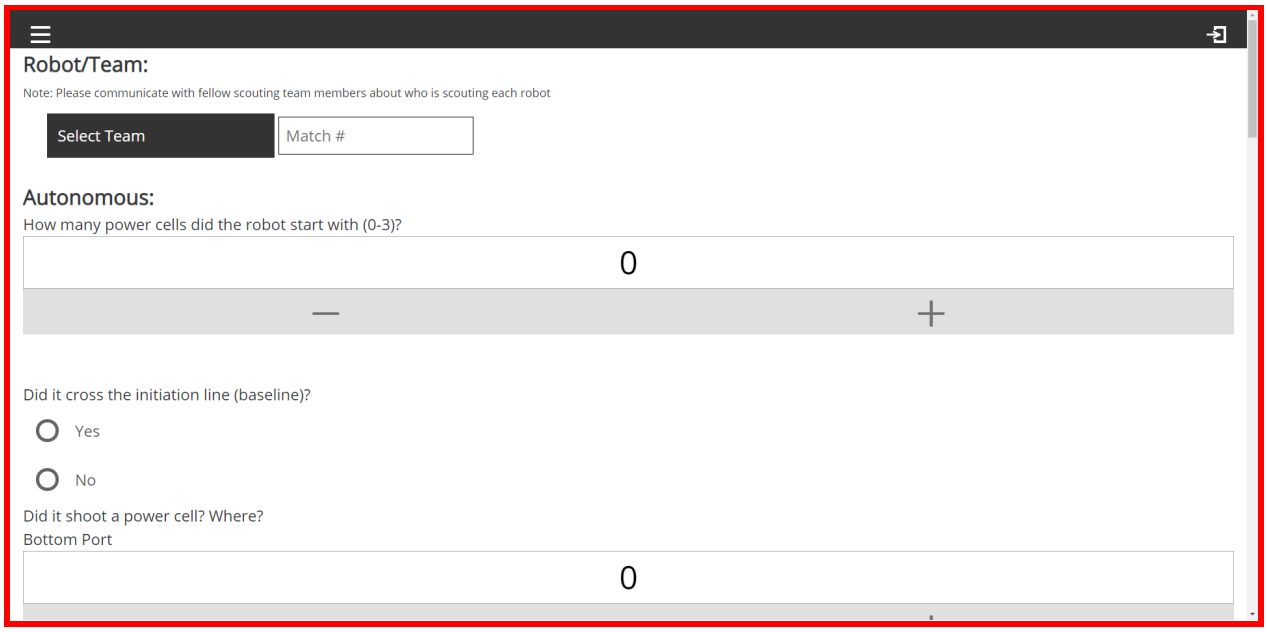
^ An image of the home page of the scouting app. On the left side you can see the open side navigation bar.

Once you get to the home page, click on the side navigation bar button in the top-left of the screen and you will see all of the different options for scouting.

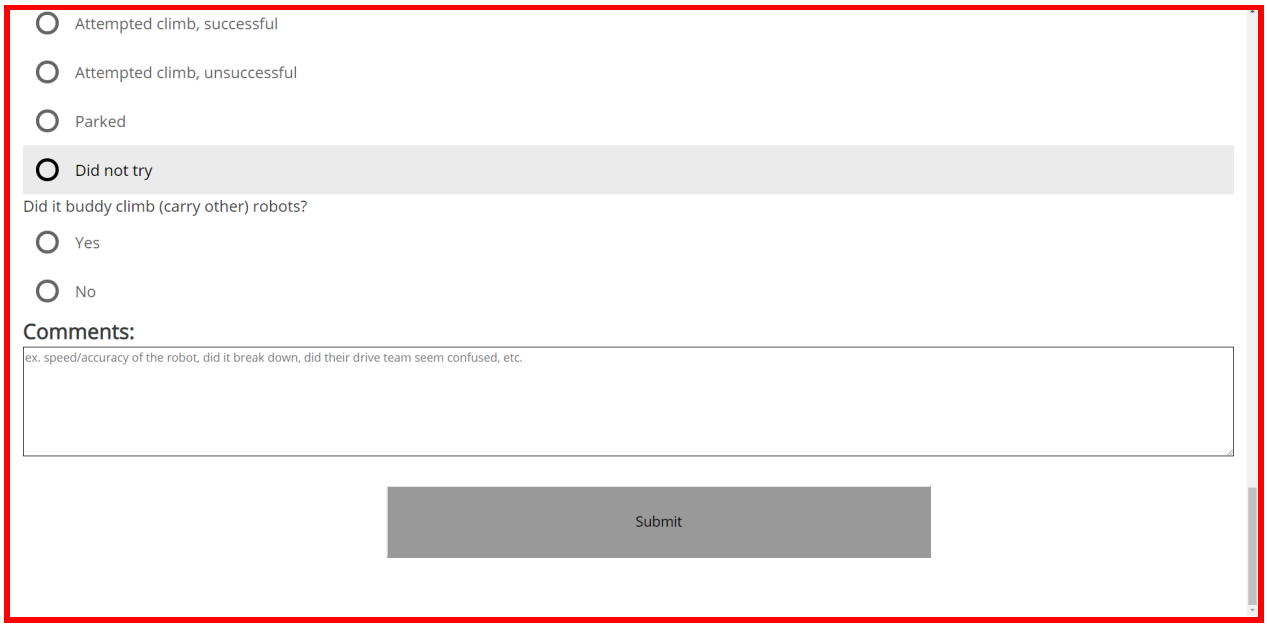
There are two main categories of scouting supported in the app: Stand Scouting (sitting and watching from the stands) and Pit Scouting (walking around in the area where teams work on their robots).

Under each category, there is a form with questions to fill out based on the type of scouting. For stand scouting, the questions are based on the performance of a team's robot during a game match. For pit scouting, the questions are based on the physical characteristics of a team's robot.

In a typical competition, pit scouting should be completed before stand scouting, prior to any qualification matches.

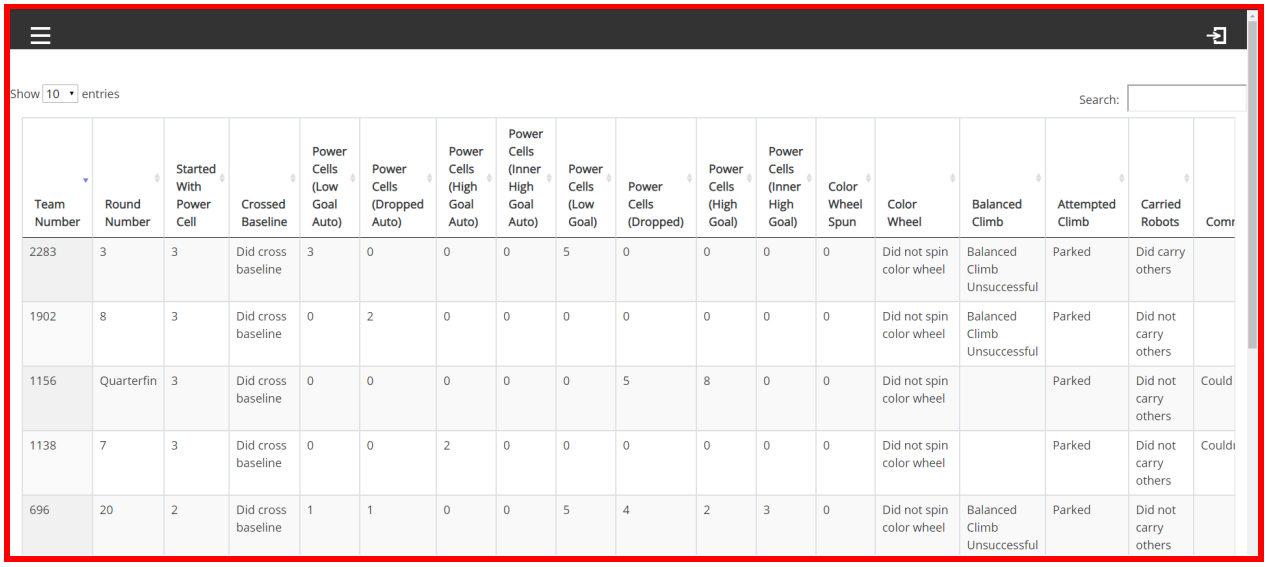


^ Image of the top of the Stand Scouting form from 2020



^ Image of the bottom of the Stand Scouting form from 2020

After the form for pit/stand scouting has been filled out and submitted, the collected data can be viewed under the 'View Data' section of the app.



Show 10 entries

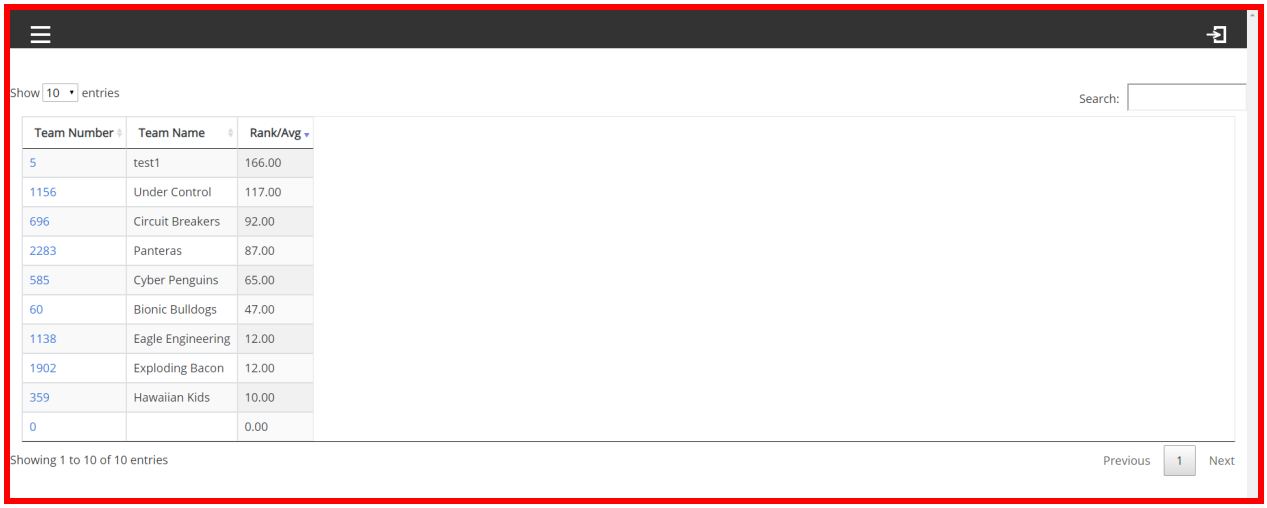
Search:

Team Number	Round Number	Started With Power Cell	Crossed Baseline	Power Cells (Low Goal Auto)	Power Cells (Dropped Auto)	Power Cells (High Goal Auto)	Power Cells (Inner High Goal Auto)	Power Cells (Low Goal)	Power Cells (Dropped)	Power Cells (High Goal)	Power Cells (Inner High Goal)	Color Wheel Spun	Color Wheel	Balanced Climb	Attempted Climb	Carried Robots	Comr
2283	3	3	Did cross baseline	3	0	0	0	5	0	0	0	0	Did not spin color wheel	Balanced Climb Unsuccessful	Parked	Did carry others	
1902	8	3	Did cross baseline	0	2	0	0	0	0	0	0	0	Did not spin color wheel	Balanced Climb Unsuccessful	Parked	Did not carry others	
1156	Quarterfin	3	Did cross baseline	0	0	0	0	0	5	8	0	0	Did not spin color wheel		Parked	Did not carry others	Could
1138	7	3	Did cross baseline	0	0	2	0	0	0	0	0	0	Did not spin color wheel		Parked	Did not carry others	Could
696	20	2	Did cross baseline	1	1	0	0	5	4	2	3	0	Did not spin color wheel	Balanced Climb Unsuccessful	Parked	Did not carry others	

^ A picture of some example data in the 'View Data' section

In the 'View Data' section, all of the data is sortable by each category. There is also a search function for the data.

Additionally, there is a 'View Ranking' section that ranks each team based on their performance across all matches.



Show 10 entries

Search:

Team Number	Team Name	Rank/Avg
5	test1	166.00
1156	Under Control	117.00
696	Circuit Breakers	92.00
2283	Panteras	87.00
585	Cyber Penguins	65.00
60	Bionic Bulldogs	47.00
1138	Eagle Engineering	12.00
1902	Exploding Bacon	12.00
359	Hawaiian Kids	10.00
0		0.00

Showing 1 to 10 of 10 entries

Previous 1 Next

^ An image of the 'View Ranking' section

Learning the Programming Languages

There are 5 programming languages you should know of, if not know how to use, when developing the scouting app: HTML, CSS, JavaScript, PHP, and SQL.

The best website that I have found to teach these languages is:
w3schools.com

This website has tutorials on all 5 languages that are used in the scouting app, and can probably teach you better than anything I can type up in this guide, but I will include a short description of what each language does and what it is used for in the scouting app.

HTML, short for Hypertext Markup Language, along with CSS, short for Cascading Style Sheets, work together to change text, background color, and other front-end parts of a website (if you don't know what "front-end" is just look it up). In other words, these 2 languages work together to change what you see on the screen.

JavaScript, serves a similar purpose as HTML and CSS, but also does a bit more for the functionality of a webpage. JavaScript can be used to create fancy animations when you click a button, display the date and time, or even do some math. In the scouting app, the main uses of JavaScript are the side navigation bar's slide-in animation, as well as the functionality of the number counters in the stand scouting form. This language is probably the least used in the scouting app, but it's still helpful to know.

PHP and SQL are languages that deal with the back-end parts of the scouting app, or in other words, areas that include the managing and storing of data. Inserting data into the database, reading and displaying the data from the database onto a webpage, and deleting data from the database are all actions that are dealt with by these 2 languages.

In my experience, the best way to learn these languages is to just jump right in and start programming, although that's easier said than done. When you program, you should have some sort of goal or idea of something that you want to make, and it's probably not the best idea to try and learn by editing a completed project made by someone else.

With this in mind, I've made a list of web-based projects with YouTube tutorials linked that I think are good for beginners:

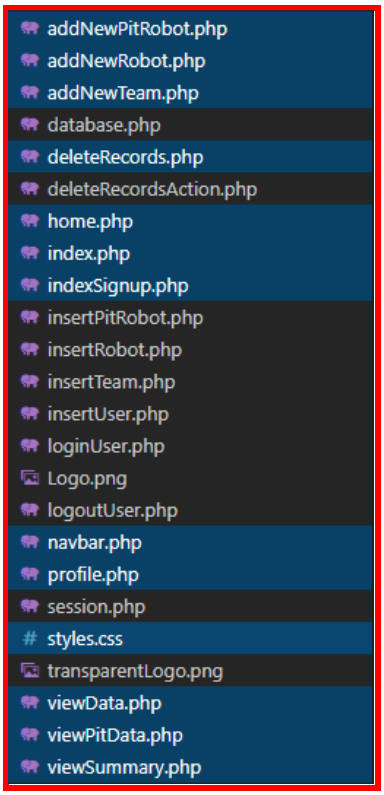
- Single Page Blog - HTML/CSS: youtu.be/wq-Q7CDj6ZI
- Login/Signup form - HTML/CSS (no back-end stuff, only front-end): youtu.be/L5WWrGMsnpw
- Login System w/ Database - HTML/CSS/PHP/SQL (has back-end stuff): youtu.be/LC9GaXkdxF8
- Basic Web Development Tutorial - HTML/CSS/JS: youtu.be/3JluqTojuME

But again, if you want to learn how to do something in these languages well, the best site that I know of is w3schools.com; they have tutorials for all 5 of these languages and more, with code examples that you can change and play around with directly on their site.

Programming the Front-End (User Interface)

Each and every web page on the scouting app has a corresponding file that dictates how that page looks and functions using HTML, CSS, and JavaScript.

These files, highlighted in the image shown below, are considered to be the 'front-end' parts of the scouting app.



^ Files highlighted in blue are the 'front-end' parts of the scouting app

To help break down the code, we will go through all of the parts of one of these files. For simplicity, we will choose the 'home.php' page, which is the first webpage that you see after you log in to the app (image on page 6).

At the very top of the file, we declare the type of document that it is (HTML) using the DOCTYPE tag. We then proceed onto Lines 3 thru 14, which is the header.

home.php

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <meta http-equiv="X-UA-Compatible" content="ie=edge">
7       <title>Firebirds Robotics Scouting</title>
8       <link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
9       <link rel="stylesheet" href="https://bootswatch.com/4/cosmo/bootstrap.min.css">
10      <link rel="stylesheet" href="https://cdn.datatables.net/1.10.16/css/jquery.dataTables.min.css">
11      <script src="https://code.jquery.com/jquery-1.12.4.js"></script>
12      <script src="https://cdn.datatables.net/1.10.16/js/jquery.dataTables.min.js"></script>
13      <link href="styles.css" type="text/css" rel="stylesheet"/>
14  </head>
15  <body class="homeBody">
16
```

^ (If it helps you keep track of things, you might note the line number on the left-hand side)

Below is a line-by-line description of everything in the header, but a general description of what we're doing in the header is setting up links, or references, to stylesheets containing CSS code that we're using.

Line-by-Line breakdown for the Header

3: The opening HTML tag for the header
4: Specifies the character encoding for the HTML document
5: Specifies a name for the metadata
6: Provides an HTTP header for the information/value of the content attribute
7: The title of the webpage (the text you see on the tab of your browser)
8: A link to the Open Sans font (so we can use the font on the webpage)
9: A link to a bootstrap theme (some open-source CSS code)
10: A link to an open-source JavaScript library called jQuery
11: Includes the open-source JS code from jQuery in our project
12: Includes a JS code that organizes the table in the "viewData" files
13: The closing HTML tag for the header

Moving on to the body, Lines 15 thru 50, you can (hopefully) recognize that we're still using HTML tags for much of our code, but there's actually some PHP code in this file.

home.php

```
14  </head>
15  <body class="homeBody">
16
17      <?php
18          include("database.php");
19
20          if (!isset($_SESSION['username'])) {
21              echo '<script type="text/javascript">location.href = "index.php";</script>';
22          }
23      ?>
24
25      <?php
26
27          include("navbar.php");
28
29      ?>
30
31      <div class="images">
32          <img src="Logo.png" alt="Logo" width=100% height=auto>
33      </div>
34
35      <div class="scoutingDescriptionAll">
36          <b>What is scouting?</b><br>
37          <div class="scoutingDescription">
38                  The purpose of scouting is to collect information about other teams' robots.
39                  With this information, we can then see how our and other teams compare to each other.
40                  This data is also useful for forming alliances in the final rounds of competitions. <br><br>
41                  There are two different categories of scouting: Pit Scouting and Stand Scouting.
42                  You can help collect information by clicking the 'Add Robot' tab in the side bar.
43                  From there you will be taken to a form to fill out with information about a specific robot.
44                  To view round data, you can click on the 'View Data' tab in the side bar.
45                  This will show you a table full of all the data collected on the robots per round.
46                  To view ranking data, you can click on the 'View Ranking' tab in the side bar.
47                  This will show you a table comparing robots/teams by average performance per round.
48          </div>
49      </div>
50  </body>
51  </html>
```

Line-by-Line breakdown for the Body

15: The opening tag for the body, declares class "homeBody"
17: Opens a block of PHP code
18: Basically copy-pastes all of the code from the 'database.php' file into this file, at Line 18
20-22: Little bit more back-end (next section), but essentially checks if the user is logged in, and if they're not, they get redirected to the login page
23: Closes the block of PHP code

25: Opens a new block of PHP code (a little redundant, I know)
27: Same as Line 18; copy-pastes all of the code from the 'navbar.php' file into this one, right at Line 27
29: Closes the block of PHP code
31: Opening tag for a 'div' section (div sections, or "dividers", are made for styling things using CSS, they don't have much other purpose)
32: Adds an image with the name "Logo.png" (you need to have the photo) with the alternative text "Logo", a width of 100% of the screen, and a height that automatically adjusts. Alternative text is what shows in place of the photo when the photo can't load
33: Closes the div section from Line 31
35: Opens a div with the class "scoutingDescriptionAll"
36: Bolds the text "What is scouting?" and creates a line break afterwards
37: Opens a div with the class "scoutingDescription"
38-47: The description of scouting (the raw text) shown on the home page
48: Closes the div from Line 37
49: Closes the div from Line 35
50: Closes the body

In Lines 17 thru 29, the code shifts from HTML to a few lines of PHP, noted by the <?php and ?> tags. The first tag begins a block of PHP code, and the second tag closes it. In the line-by-line breakdown, you can look at what each line of PHP code is doing in the project.

You'll also notice that we have multiple words in each tag; notably the word 'class' with a string following it. The 'class' keyword defines the class of that block of HTML code, and in the CSS file ("styles.css") we can reference that class to change how things look for everything inside the HTML tags that have that specified class.

```
# styles.css > ...
354    .homeBody{
355        background-color: #000000;
356        color: #ffffff;
357    }
```

^ An image of the CSS for the class "homeBody".

Here, the CSS of the class "homeBody" changes the background color of elements within the HTML tags with the class "homeBody" to black, and the text color of elements within those tags to white.

Now that we've covered how HTML and CSS work together in the scouting app, there's still the language of JavaScript. I mentioned earlier in this guide that JavaScript is probably the least used language in the scouting app, and that's because it's only used in 2 places: 'navbar.php' and 'addNewRobot.php'.

In 'navbar.php', we use JavaScript to make a nice little slide animation for the side navigation bar, as shown below:

```
navbar.php
47      <script type="text/javascript">
48          //Sidebar
49      function openSlideMenu(){
50          document.getElementById('side-menu').style.width ='250px';
51          document.getElementById('main').style.marginLeft ='250px';
52      }
53
54      function closeSlideMenu(){
55          document.getElementById('side-menu').style.width ='0';
56          document.getElementById('main').style.marginLeft ='0';
57      }
58      </script>
```

From this picture, you can see that the tags for using JavaScript inside an HTML doctype are simply <script type="text/javascript"> and </script>

In 'addNewRobot.php', we use JavaScript to create a working number counter like so:

addNewRobot.php

```
242     <script type="text/javascript">
243         // Number Counter Code
244     function modifyQty(val) {
245         var qty = document.getElementById('qty').value;
246         var new_qty = parseInt(qty,10) + val;
247 
248         if (new_qty < 0) {
249             new_qty = 0;
250         }
251 
252         document.getElementById('qty').value = new_qty;
253         return new_qty;
254     }
```

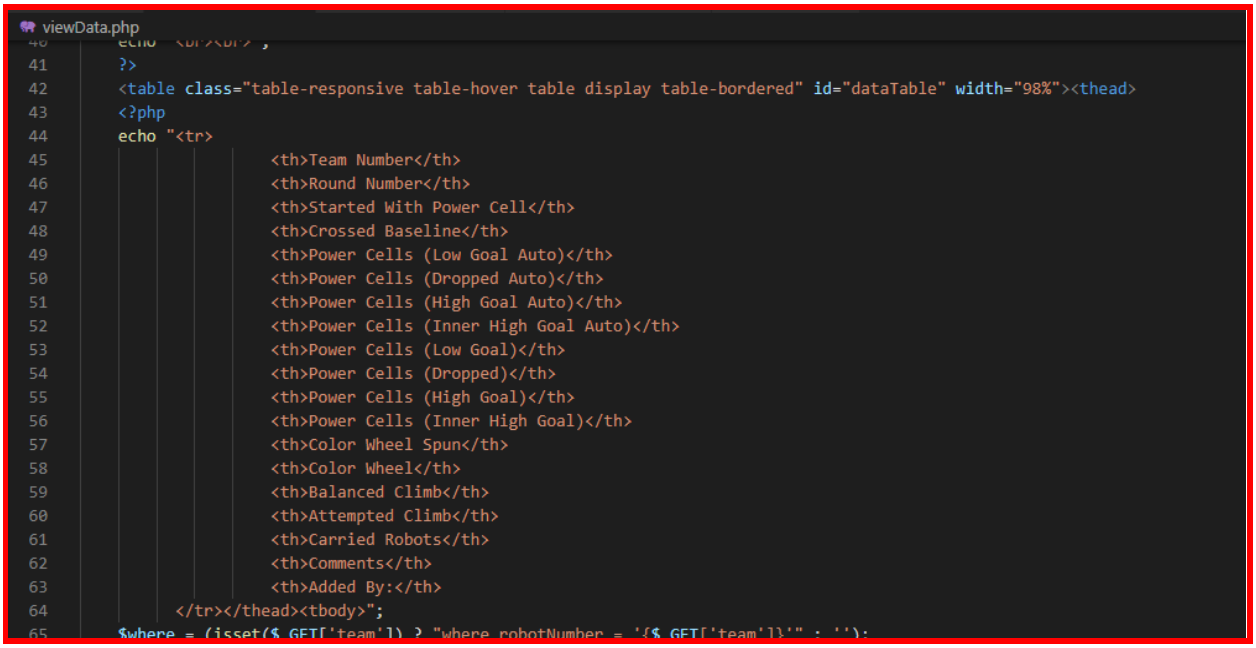
The JavaScript function "modifyQty" (shown above) finds an HTML element with the ID "qty" and changes its value using some simple logic. The function is called by buttons made in HTML that have the onclick attribute (shown below).

addNewRobot.php

```
115         <b><font size="+3">Teleop:</font></b><br>
116             How many power cells did it shot in the bottom port?<br>
117             <div class="numCounter">
118                 <input class="qty" id="qty" value="0" name="lowGoal"/>
119                 <button class="redButton counterButton" id="sub" onclick="modifyQty(-1); return false;">-</button>
120                 <button class="greenButton counterButton" id="add" onclick="modifyQty(1); return false;">+</button>
121             </div><br><br><br><br>
122 
```

The rest of the "front-end" files contain much of the same content as the 'home.php' file, with the big exception being the 'viewData.php', 'viewPitData.php' and 'viewSummary.php' files. These 3 files have more to do with PHP than HTML, but they're still considered front-end pages.

We won't go over these files line-by-line, but there is one thing that I want you to note — look at Lines 45 thru 64 of 'viewData.php'.



In these lines (as well as in the lines following these, later in the file), we are actually writing HTML code inside of a PHP echo statement (echo is like a print statement in PHP). It is important that you realize that this is possible, and that it is something we can use to our advantage.

There are likely lots of HTML tags used in the scouting app that we didn't cover in this guide, but you don't need to know every single tag to understand what is going on in the front-end side of things.

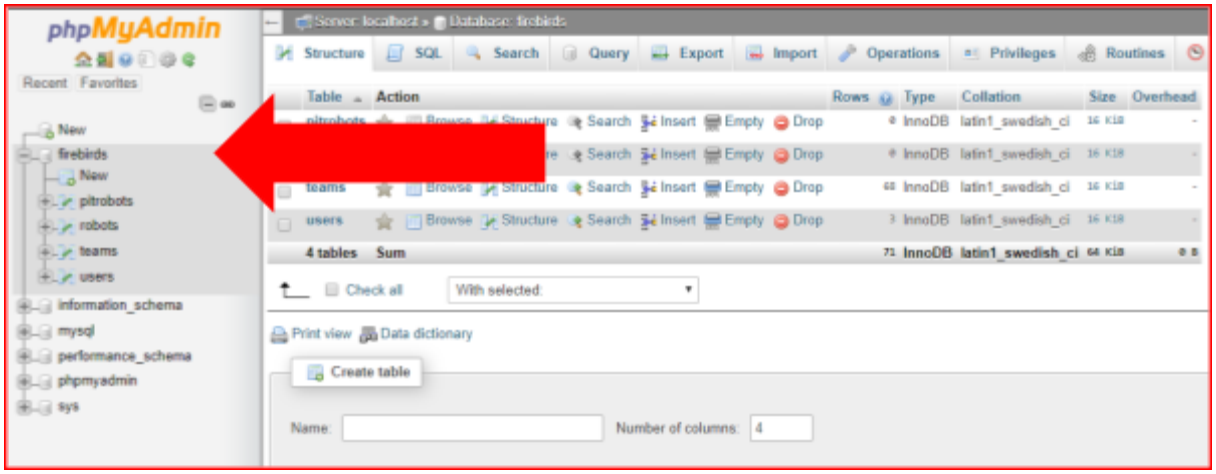
Remember, you can always look up something that you don't know (and I suggest that you do), especially if it's just something like an HTML tag that you don't know. As long as you have a basic understanding of how HTML code works and looks, that's good enough for the sake of this guide.

Programming the Back-End (Data & Database)

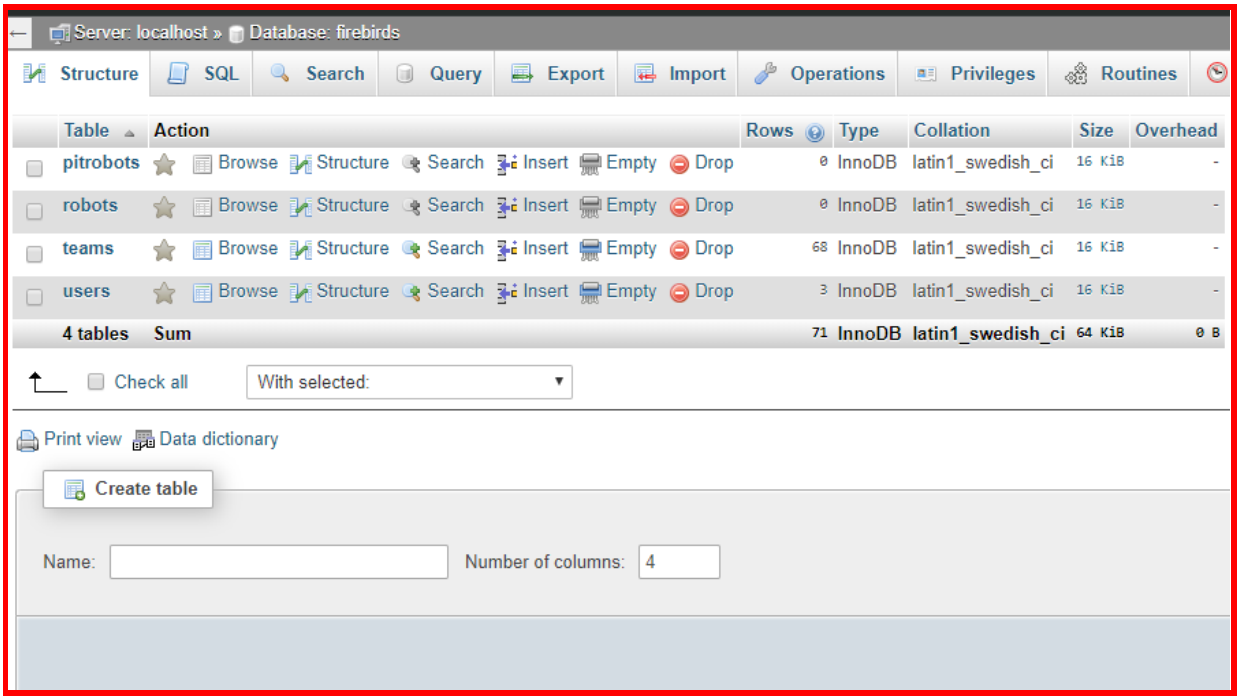
The back-end section of the scouting app might seem a little bit more complicated than the front-end, and that's because it is. We'll first look at the actual database in a program called PHPMyAdmin, and then we'll go over the '.php' files associated with the back-end scene of things.

No matter where you're trying to view the database of the scouting app, you'll need to find your way to a program called PHPMyAdmin. The ways to find this program by setting up a local web server hosted on your computer, or by navigating through your online hosting service are described in the following sections, so for now we'll just talk about how to view your database once you have access to the program and everything is set up.

After you log in to PHPMyAdmin (also described in the next section), you'll see a side navigation bar on the left-hand side. There will be an item in the navigation bar called 'firebirds'. That is the name of our database.



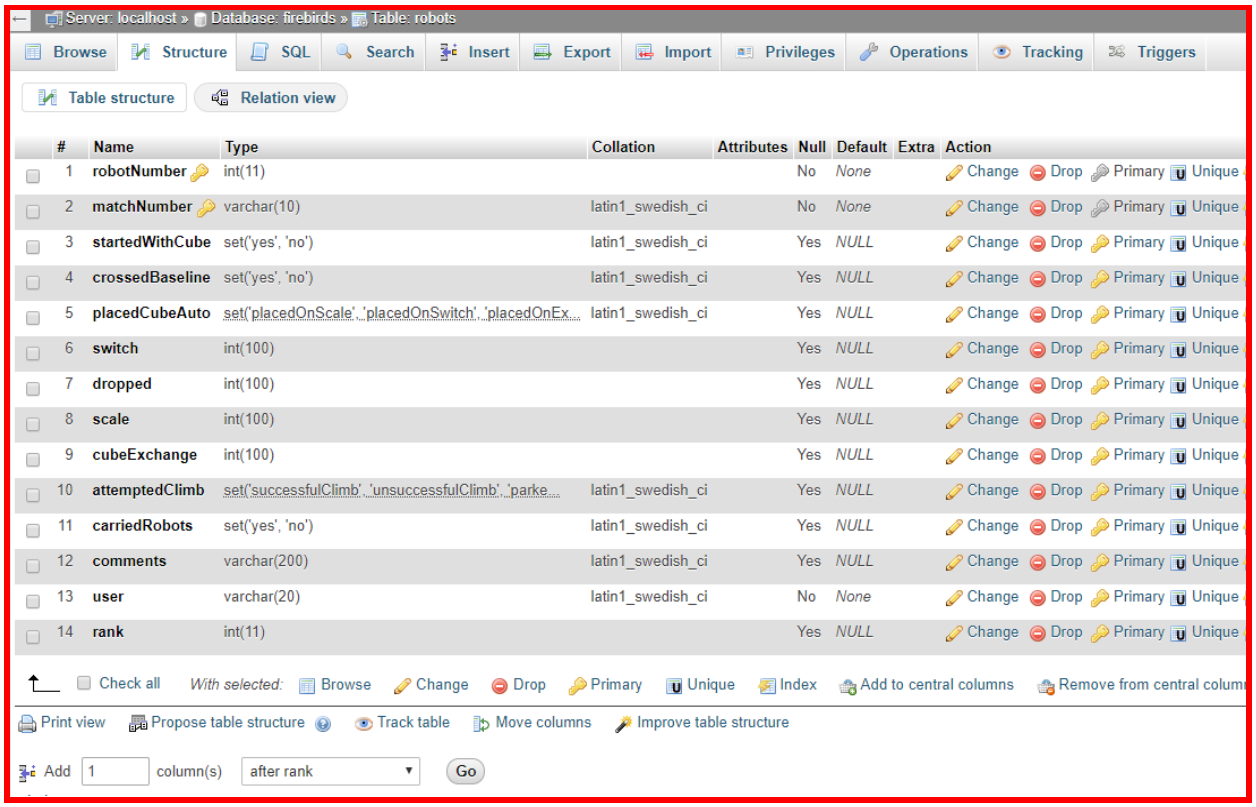
Upon clicking the database, there will be a list of 'Tables' that show up on both the navigation bar, and in the center of your screen. These tables are the same as the ones we display on the 'View Data' screen of the app.



^ The list of tables in our database. You can see we have 4 tables, called 'pitrobots', 'robots', 'teams', and 'users'.

If you click on one of the tables in the database, such as 'robots', you can see all of the different columns of the table, as well as all of the data that has been entered into the table.

Note: This also means that anyone who has access to the database through PHPMyAdmin can see all of the passwords and user information in the 'users' table, so it's very important that the database is secure; Luckily, the website hosting company that we use, GoDaddy, does this for us (read more about this in the next section).



^ An image showing the different columns of the table 'robots' in the database from the 2018 game

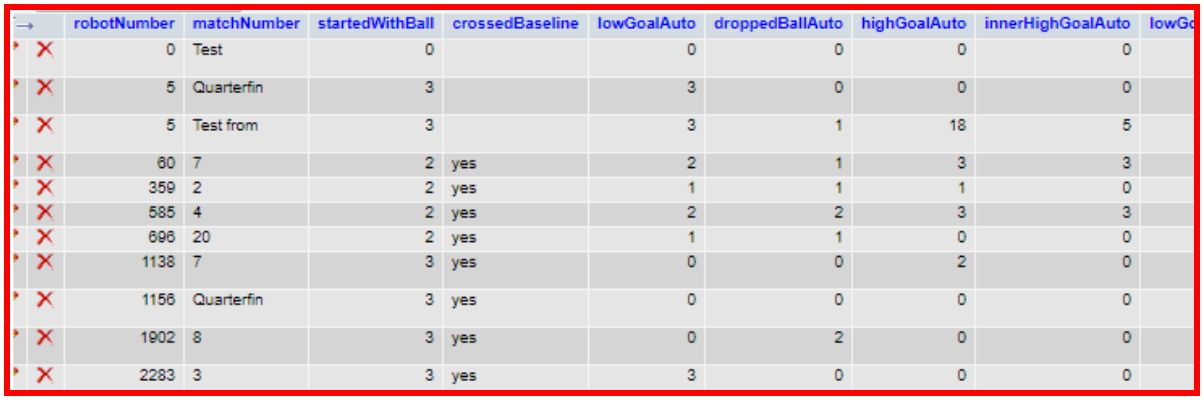
In the above image, you can see that there are 14 different entries under the 'Name' category: 'robotNumber', 'matchNumber', and so on. These are how the data is organized, and are the columns of our data table — for each question in our 'Add Robot' form, there is a corresponding column in our data table.

You'll also notice that to the right of each label, under the 'Type' category, there is something different for each column. These are the acceptable types of data for each column of our table. Below is a list of the ones seen in the image:

- int: restricts the data type to a whole number
- varchar: allows only letters and numbers
- set: allows a few specific preset values (those in the parenthesis)

As for the rest of the categories of our table, you can ignore the 'Collation' and 'Attributes' categories, but note the 'Null' and 'Default' categories (they kinda go hand-in-hand) — if the 'Null' category says 'No' for a column, then that means there must be a value for that column. If it says 'Yes', it means that there isn't a value required for that column.

The 'Default' category is the default value for a table entry if one of the columns is left blank; since the columns that require a value can't be empty, they don't have anything in the 'Default' category.



	robotNumber	matchNumber	startedWithBall	crossedBaseline	lowGoalAuto	droppedBallAuto	highGoalAuto	innerHighGoalAuto	lowGo
	0	Test	0		0	0	0	0	
	5	Quarterfin	3		3	0	0	0	
	5	Test from	3		3	1	18	5	
	60	7	2	yes	2	1	3	3	
	359	2	2	yes	1	1	1	0	
	585	4	2	yes	2	2	3	3	
	696	20	2	yes	1	1	0	0	
	1138	7	3	yes	0	0	2	0	
	1156	Quarterfin	3	yes	0	0	0	0	
	1902	8	3	yes	0	2	0	0	
	2283	3	3	yes	3	0	0	0	

^ A snippet of some test data in the 'robots' table from the 2020 game

In the above image, you can see what some sample data looks like in the database (it's a bit zoomed in and cut-off since otherwise you can't read the text). Notice how the first 2 columns match the data types mentioned earlier — the robotNumber column only has numbers, while the matchNumber column has both letters and numbers.

Now that you have a decent idea of what the database end of things looks like, we'll start looking at connecting the database to the front-end of things through code, specifically through PHP and SQL.

The first thing that we want to do is be able to put data into the database. To do this, we need to:

1. Gather the data in an HTML form
2. Create a connection to the database
3. Insert the data from the HTML form into the database

So let's look at the HTML form, in the file 'addNewRobot.php'. You can spend time exploring the rest of the file on your own, but right now let's focus on the code starting on line 36:

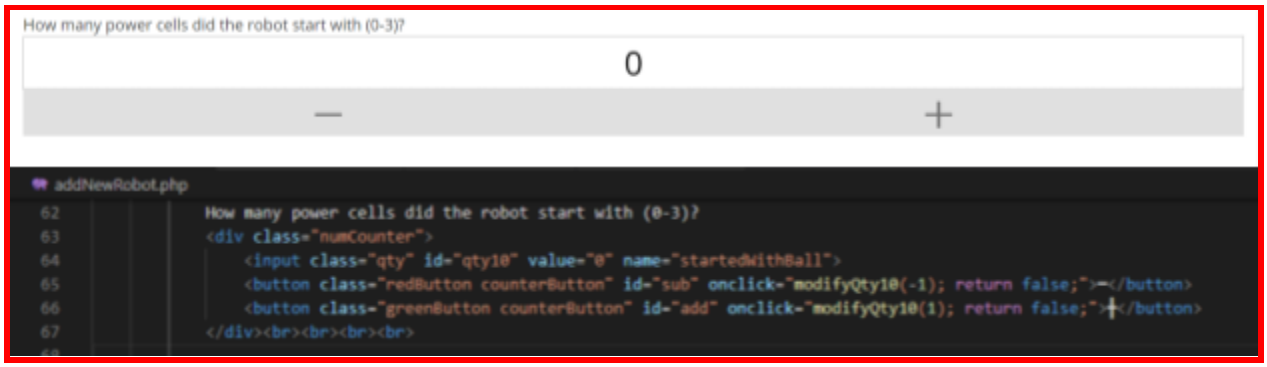
addNewRobot.php

```
36          <form action="insertRobot.php" method="post">
```

This line is the opening tag for our HTML form. It has an action of "insertRobot.php", which means that after the form is submitted, it will redirect the user to the 'insertRobot.php' file (and go through all of the code in that file). It also has a method of "post", which tells the web browser to send the data to the web server for it to be processed.

Inside of the form, we have all of the HTML code for the questions in the 'Add Robot' form. There are a number of different types of questions in the form, so we will look at each type of question along with its corresponding HTML code in a side-by-side image.

The first question type I want you to look at are the questions with the number counters. The following image shows the actual webpage question on top, with the HTML code on the bottom.



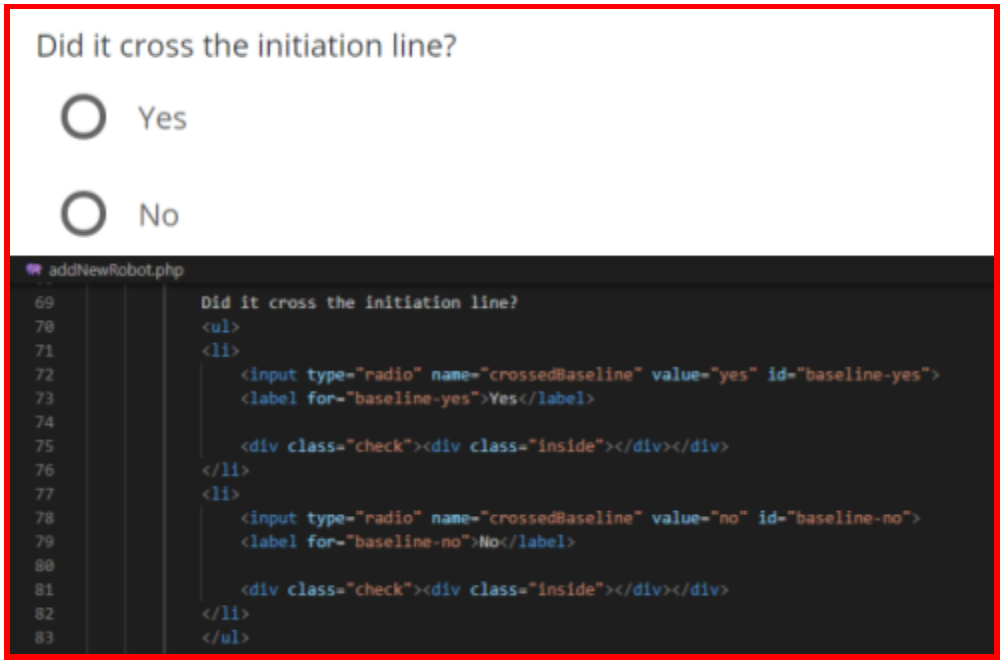
^ The form question 'startedWithBall' from the 2020 game

Here, you can see that the number counter is inside a div called "numCounter", and that there are 3 things inside of that div. The <input> tag is the text field that displays the number, and the 2 <button> tags are the buttons below the text field (one with a "-" sign, the other with a "+" sign).

The input, which actually holds the value for this question, has a name of "startedWithBall", which corresponds to the column "startedWithBall" in our database table.

What you need to gather from this is that each question has a name that corresponds to a column of our data table, and that it submits a value for that column once the form is submitted.

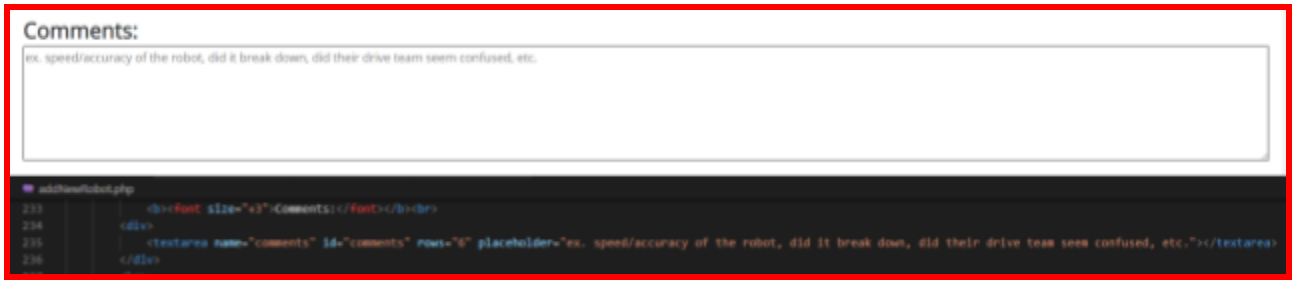
Next we'll look at the radio button questions, which you might know better as multiple-choice questions.



You can look up what the <ul> and <li> tags do (as you can with all HTML tags), but just know that they create that multiple-choice format.

You can see that both of the radio buttons (marked by the <input> tag with the type of “radio”) have a name of “crossedBaseline” (corresponding to the column of our database table), as well as the fact that one of the buttons has a value of “yes”, and the other has a value of “no”.

Finally, we’ll look at the question with a text box, the comment box.



Here, we make the text box object using the <textarea> tag.

Again, you can see that we have a name element, in this case with a name of “comments”. However, the text box is a bit of an exception from the other questions — it does not have a value, since its value is whatever is in-between the <textarea> tags.

addNewRobot.php

```
237        <br>
238            <input type="submit" value="Submit">
239        </form>
240        <br>
```

^ The HTML for the Submit button at the bottom of the ‘Add Robot’ form

Right at the end of the form, before we close it off, we have one last <input> tag, with a type of “submit”. Pressing this button will submit the form, redirecting the user to the ‘insertRobot.php’ file and causing the web browser to send the data to the web server (due to what we put in the opening tag of our form).

Inside of ‘insertRobot.php’, we will do a few crucial things needed in order to push data into the database.

Note: Put simply, ‘addNewRobot.php’ collects the data, while ‘insertRobot.php’ puts the data into the database. As you’ll see a bit later, ‘viewData.php’ displays the data from the database. These 3 files are sort of like the core functionality of the scouting app; almost everything else is more or less a copy of one of them.

insertRobot.php

```
6
7   <?php
8
9       include("database.php");
10
11      // Get user who added
12      $sqlForUser = "SELECT * FROM users where username = '". $_SESSION["username"] ."'";
13      $resultForUser = $conn->query($sqlForUser);
14      $rowForUser = mysqli_fetch_assoc($resultForUser);
15      extract($rowForUser);
16
17      $user = "$rowForUser[firstName] $rowForUser[lastName]";
18
```

^ The beginning few lines of the PHP code in the 'insertRobot.php' file

As you can see on Line 9, the first thing that we do is include (copy-paste) the 'database.php' file. This file (shown below) creates a connection to the database using PHP and SQL.

database.php

```
1   <?php
2
3       $servername = 'localhost';
4       $username = 'root';
5       $password = 'root';
6       $dbname = 'firebirds';
7
8       // Create connection
9       $conn = new mysqli($servername, $username, $password, $dbname);
10
11      // Check connection
12      if (mysqli_connect_errno()) {
13          echo "Failed to connect to MySQL: " . mysqli_connect_error();
14      }
15
16      session_start();
17  ?>
```

^ The database.php file (Yes, it is 17 lines and that's it)

On Line 9, we create a connection to the database using mysqli() (I would suggest looking up "mysqli construct") and then assign it to the PHP variable $conn, which we use in our other files.

Another important thing that we do in 'database.php' is start what is called a session — sessions store data for individual users via session cookies (you may have heard of cookies before) and differentiate between one person and another. Starting a session is also necessary to send data.

Now that we've covered 'database.php', we'll continue in 'insertRobot.php'.

insertRobot.php

```
11      // Get user who added
12      $sqlForUser = "SELECT * FROM users where username = '". $_SESSION["username"] ."'";
13      $resultForUser = $conn->query($sqlForUser);
14      $rowForUser = mysqli_fetch_assoc($resultForUser);
15      extract($rowForUser);
16
17      $user = "$rowForUser[firstName] $rowForUser[lastName]";
18
19      // Add variables
20      $robot_number = mysqli_real_escape_string($conn, $_POST['teamNumber']);
21      $match_number = mysqli_real_escape_string($conn, $_POST['matchNumber']);
22      $started_with_ball = mysqli_real_escape_string($conn, $_POST['startedWithBall']);
23      $crossed_baseline = mysqli_real_escape_string($conn, $_POST['crossedBaseline']);
24      $low_goal_auto = mysqli_real_escape_string($conn, $_POST['lowGoalAuto']);
25      $dropped_ball_auto = mysqli_real_escape_string($conn, $_POST['droppedBallAuto']);
26      $high_goal_auto = mysqli_real_escape_string($conn, $_POST['highGoalAuto']);
27      $inner_high_goal_auto = mysqli_real_escape_string($conn, $_POST['innerHighGoalAuto']);
28      $low_goal_ball = mysqli_real_escape_string($conn, $_POST['lowGoal']);
29      $drop_ball = mysqli_real_escape_string($conn, $_POST['dropped']);
30      $high_goal_ball = mysqli_real_escape_string($conn, $_POST['highGoal']);
31      $inner_high_goal = mysqli_real_escape_string($conn, $_POST['innerHighGoal']);
32      $color_wheel_spun = mysqli_real_escape_string($conn, $_POST['colorWheelSpun']);
33      $color_wheel = mysqli_real_escape_string($conn, $_POST['colorWheel']);
34      $balanced_climb = mysqli_real_escape_string($conn, $_POST['balancedClimb']);
35      $attempt_climb = mysqli_real_escape_string($conn, $_POST['attemptedClimb']);
36      $carry_robots = mysqli_real_escape_string($conn, $_POST['carriedRobots']);
37      $comments = mysqli_real_escape_string($conn, $_POST['comments']);
38      $user = mysqli_real_escape_string($conn, $user);
39      $rank = 0;
40
```

In Lines 12 thru 17, we go through a process to find the user that submitted the data — first we find which row the user is in (in the 'users' table of our database) by searching with their username, and then using that row to get their first and last names; This data is shown on the last column of the table in our 'View Data' page.

The next thing that we do, in Lines 20 thru 39, is create variables for each of the collected pieces of data and turn those pieces of data into strings (mysqli_real_escape_string does this). You'll notice we use the $conn variable, which we created in 'database.php', as well as the same thing in the HTML name element of each of the form questions.

Finally, we insert the data into our database. Lines 43 and 44 assign some SQL code to a variable $sql, which we then call in Line 47 (inside of the if-statement). This SQL code essentially takes each value of the 'Add Robot' form (the variables we assigned above) and puts it into the table 'robots'. If there are any entries that have the same 'robotNumber' AND 'matchNumber' (names of the columns in our database table), they will be replaced, indicated by the REPLACE keyword on Line 43.

insertRobot.php

```
42      // Insert the above variables into the table values
43      $sql="REPLACE INTO robots (`robotNumber`, `matchNumber`, `startedWithBall`, `crossedBaseline`, `lowGoalAut
44                  VALUES ('$robot_number', '$match_number', '$started_with_ball','$crossed_baseline', '$low_
45
46
47      if ($conn->query($sql) === TRUE) {
48          echo 'Robot successfully added' . '<br><a href="viewData.php">Click here to view data</a>';
49      } else {
50          echo "Error: " . $sql . "<br>" . $conn->error;
51      }
52
53      echo '<br><br>';
54
55      $conn->close();
56
57      ?>
58
```

If you're having a hard time understanding how the SQL code is called, I would suggest looking at the w3schools tutorial for the PHP mysqli query() function:

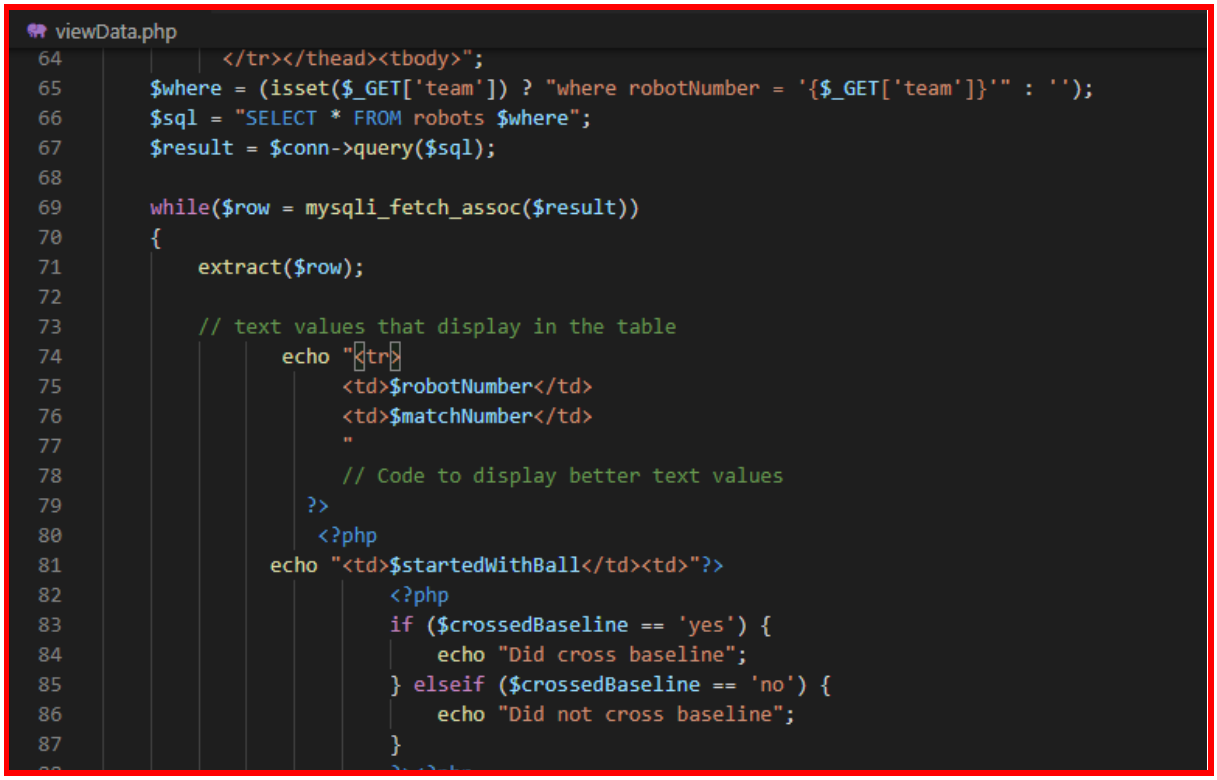
www.w3schools.com/php/func_mysqli_query.asp

Afterwards, we simply close the connection on Line 55 (since we're no longer putting things into the database).

Note: One small note I want to make about 'insertRobot.php' (along with the other 'insert' files) is how at the very top, on Line 3, it makes use of an HTML trick to immediately redirect the user to 'viewData.php'. When debugging any changes to the scouting app (meaning, when you're having issues with your code), it's a good idea to just comment out this line of code (ctrl + /) so that you can see any errors you might have.

After being able to insert data into the database, we need to be able to view all of the data in the table. We can see how this is done by looking at the 'viewData.php' file.

We've already gone over a small part of this file in the Front-End section of the guide, but now we want to look at the part that uses the database.



In Line 25 of this file (not shown), we include 'database.php' and create a connection to the database. In Lines 65 thru 67 (shown), we go through a process (assigning multiple variables) to get each row of the table, and assign it to the value $result.

The way we ensure we get ALL teams in the table is through the $where variable in Line 65, which says: If we can find a value 'team' in the table, then let $where equal "where robotNumber = '{$_GET['team']}'", otherwise if we can't find a value 'team', then let $where equal ' ' (nothing).

So what we're doing in Line 65 is actually what is called an inline if-statement (it's a good idea to learn more about inline statements since they appear in a lot of places — not necessarily just the scouting app, but in all code).

And then finally in Line 69 & beyond, we make a while loop that says: while the variable $result exists (has a value), go through the following code (which populates the table)

One final thing that I want to show in this file is the following snippet:

viewData.php

```
81                  echo "<td>$startedWithBall</td><td>"?>
82                      <?php
83                      if ($crossedBaseline == 'yes') {
84                          echo "Did cross baseline";
85                      } elseif ($crossedBaseline == 'no') {
86                          echo "Did not cross baseline";
87                      }
88                      ?><?php
89                  echo "</td>
```

It might look a little bit messy, but in Lines 83 thru 87, all we're doing is changing the displayed text based on the value of the returned $crossedBaseline variable (which we got from the database). If $crossedBaseline returns a value of 'yes', rather than displaying that exact string, which is all lowercase, possibly confusing in the context of the table, etc., we display a different value that looks cleaner and makes more sense. And we also do the same thing in the case that it returns 'no', but with a different message. We use this method multiple times in the file, not just with $crossedBaseline, but also with all of the other variables that have a type of set() in the database.

If you're feeling a little bit overwhelmed by all of this, I'd recommend that you watch this short (3 minute) video on displaying data from a database using PHP and SQL:

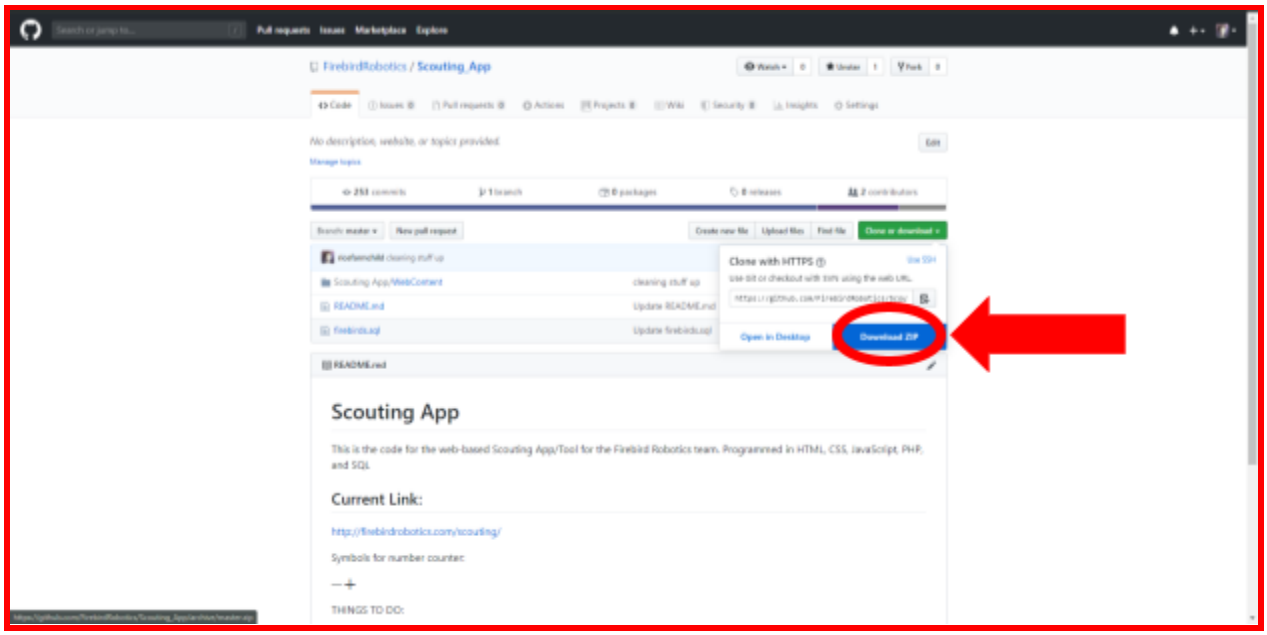
youtu.be/bHFoobciCTM

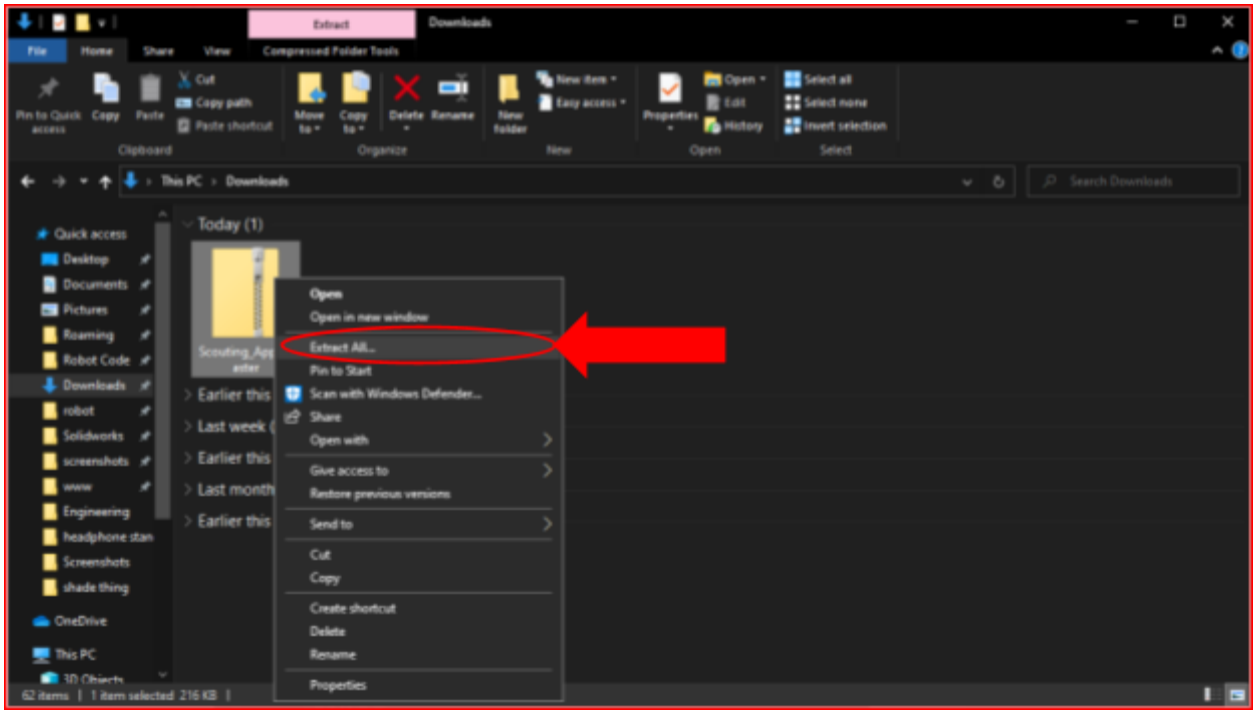
Modifying & Making Changes to the Scouting App

If you are trying to work on the scouting app, this section of the guide is to show you how to set up a working test environment to do so. The following is a list of steps (with pictures) that you'll need to follow to get everything set up correctly.

Step 1: Downloading the code from GitHub

To modify the code, you first need to download it from GitHub at this link: github.com/FirebirdRobotics/Scouting_App as a .zip file, and then extract the .zip into your downloads folder (or wherever).





This download contains all of the '.php' files and logo images for the Scouting App in the 'WebContent' folder, as well as the SQL database in the file called 'firebirds.sql'.

Step 2: Setting up a 'localhost' server and database on your computer

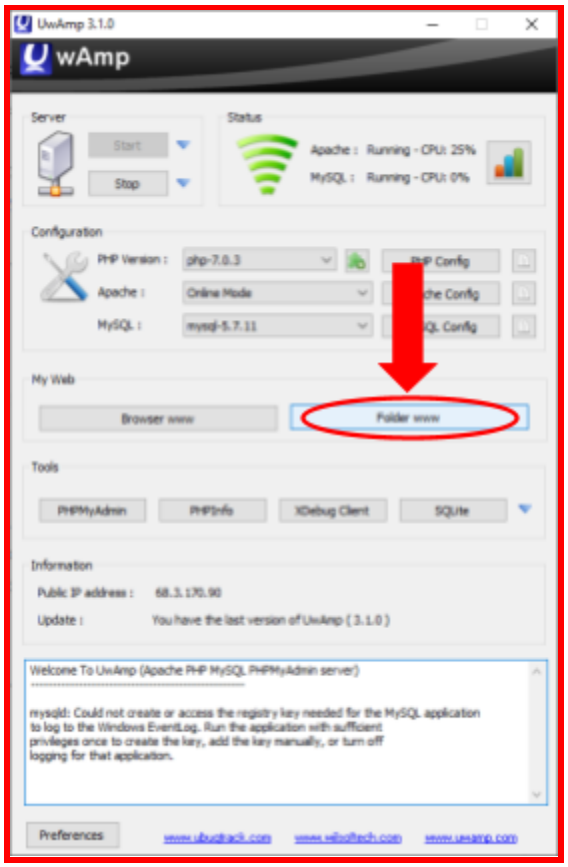
Since you can't access the actual database that's online (and it's probably not a good idea to experiment on the actual live scouting app), you need to download a program that can run a server and a database locally.

There are numerous programs that can do this (WAMP, XAMPP, MAMP), but the one that I recommend (the one I am most familiar with) is called UwAmp, which can be downloaded at www.uwamp.com.

Simply follow the installation instructions and then open up the program.

After opening UwAmp, you will have started a local server, which can be seen by opening any browser and typing in 'localhost'. However, you don't have any HTML code in your local server, so we're going to put the scouting app code up.

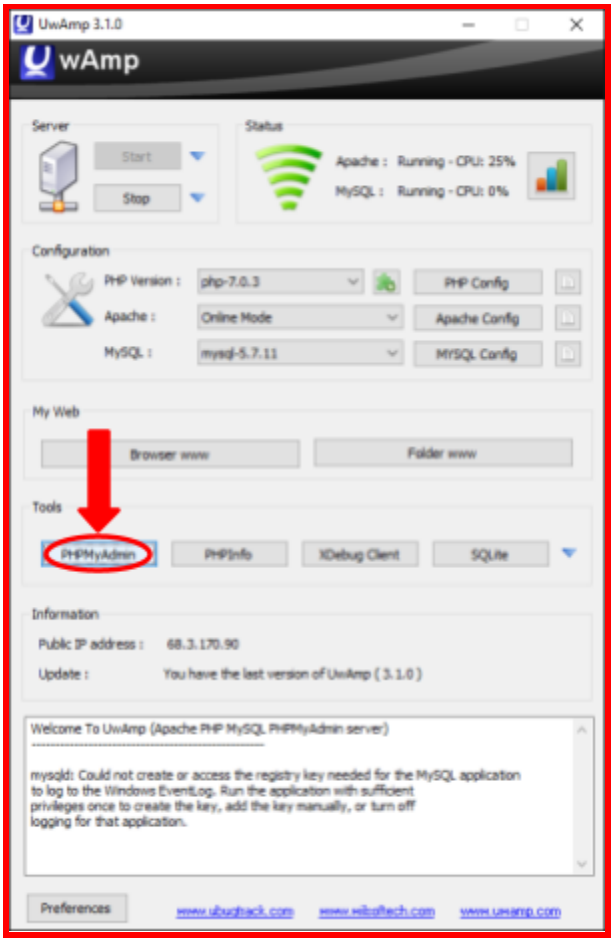
In the UwAmp app, click the button that says 'Folder www'. It will bring you to a File Explorer window that should have 1 file called 'index.html' in it.



Drag all of the scouting app code that you downloaded (in the 'WebContent' folder) into the 'www' folder, and if it says there is already a file name 'index', replace it with the '.php' one that you downloaded. If it doesn't ask you to replace the file, you can just delete the 'index.html' one that was already there.

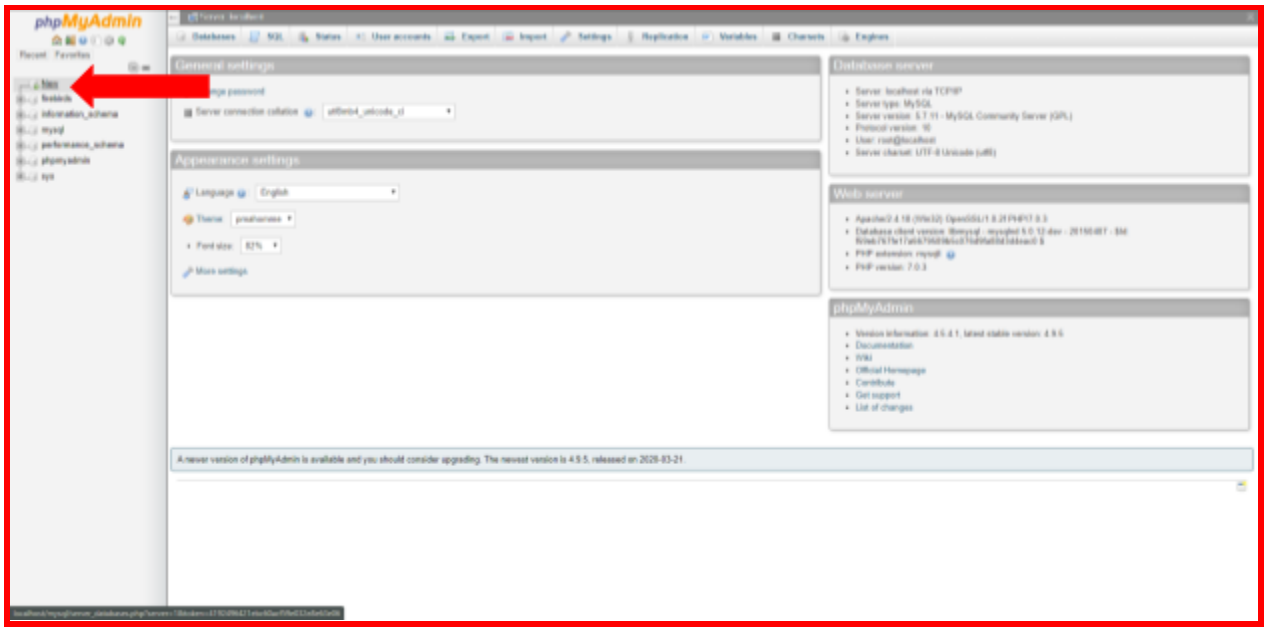
If you try to access 'localhost' now, you should see the scouting app code. However, none of the functionality will work (logging in, signing up, etc.), since you don't have the database set up.

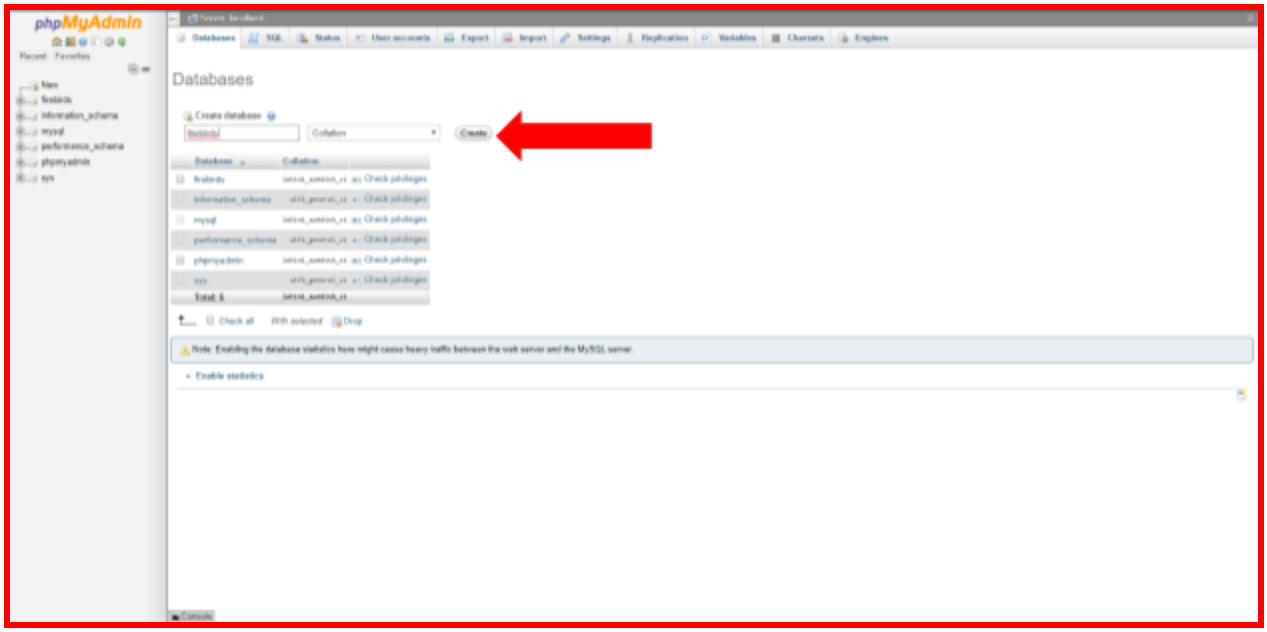
To set up the database, go back to the UwAmp application and click on the button that says 'PHPMyAdmin'.



This will bring you to the PHPMyAdmin tool, where you need to login. The default username is 'root', with the password 'root' (4 letters, all lowercase).

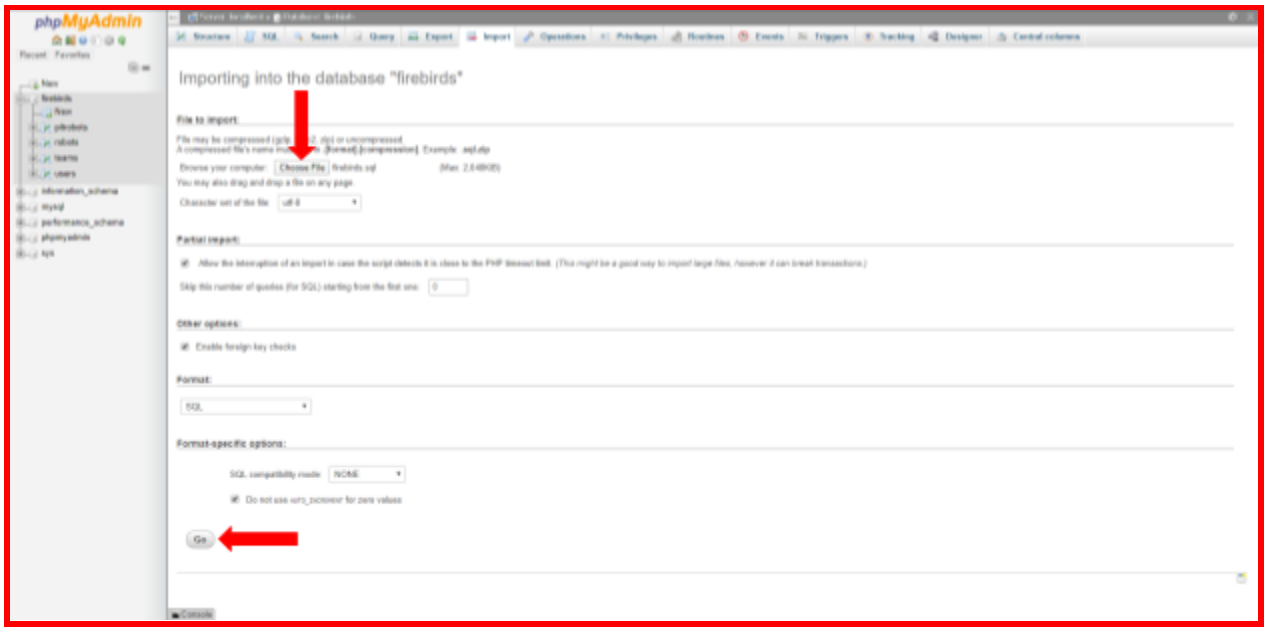
Once in the PHPMyAdmin tool, you'll need to add a new database and name it 'firebirds' (it must be named exactly like this).





After you've created the database, you need to import the tables that we use in the scouting app. Click on your newly created 'firebirds' database and then click the 'Import' button at the top.

Under the 'File to Import' area, either drag and drop or select the 'firebirds.sql' file that you downloaded from GitHub, and then press the 'Go' button at the bottom of the screen.



This should automatically import the tables for the scouting app, so after this is done, you can close PHPMyAdmin, and the scouting app should be working properly on 'localhost'.

Step 3: Setting up an environment for coding

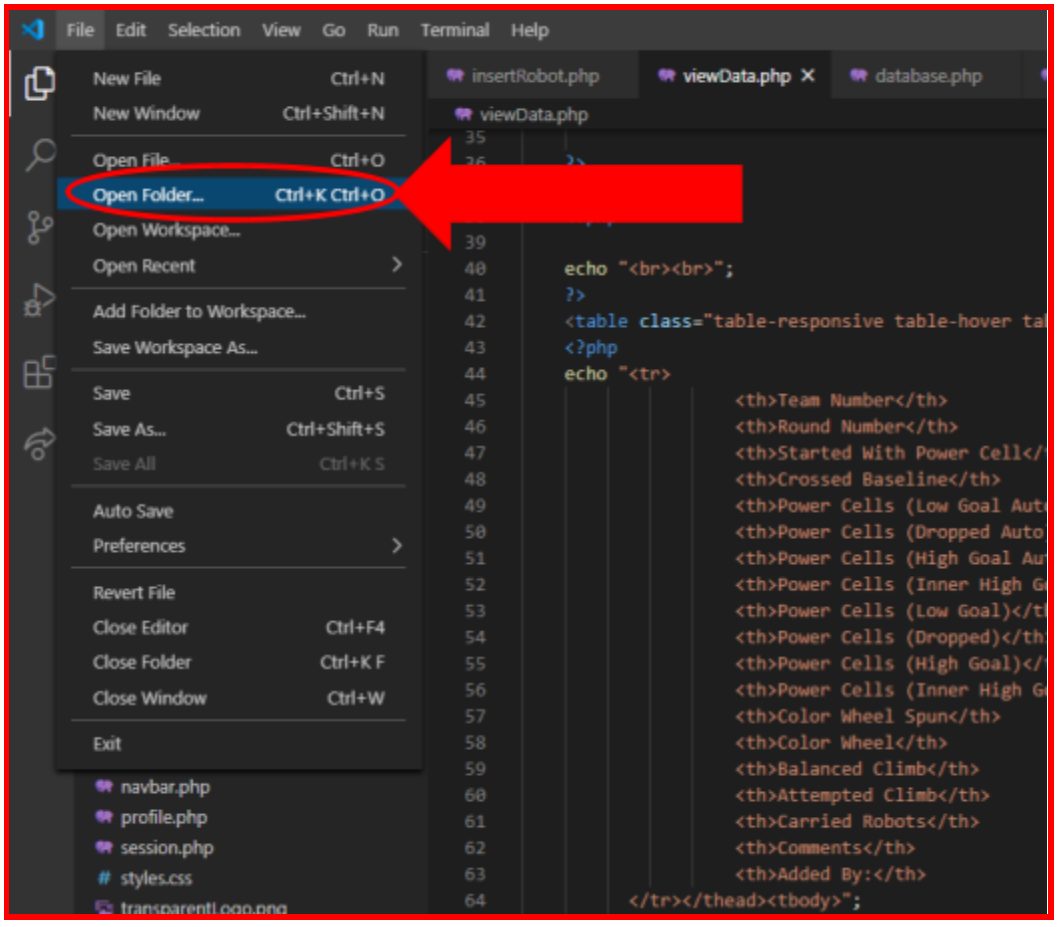
To make changes to the scouting app, you could technically just open up one of the '.php' files in Notepad and start changing things, although this wouldn't be very easy to do nor would it be much fun.

To make changing code easier and more visually appealing, I recommend that you set up what is called an IDE — an Integrated Development Environment — or essentially a program to work with code in.

There are tons of different IDEs that are widely used, but if you're working on a school computer, I recommend using either Brackets or Visual Studio Code. Some other ones that I've used before are Eclipse, Sublime Text, and Atom, but for the sake for this guide, we'll go over how to install Visual Studio Code, or VSCode for short.

To download the program, visit code.visualstudio.com and follow the installation instructions.

After you set up and open the program, go to File > Open Folder and search for the 'www' folder from UwAmp in File Explorer.



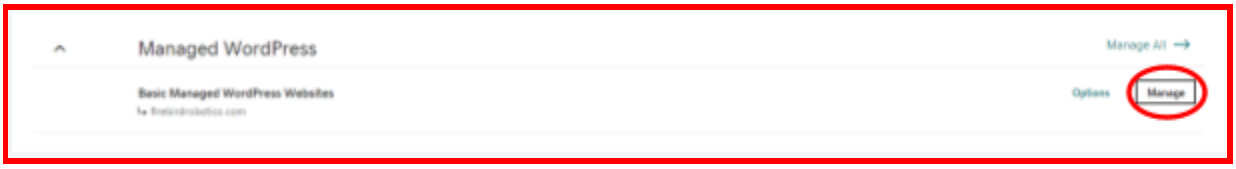
After you open the folder, you should be able to see all of the '.php' files on the left-hand side of your IDE. Just double-click on one to open it and you can start playing around with the code.

Just remember to save the file in the IDE once you're done editing and reload the localhost page in your browser to see your changes!

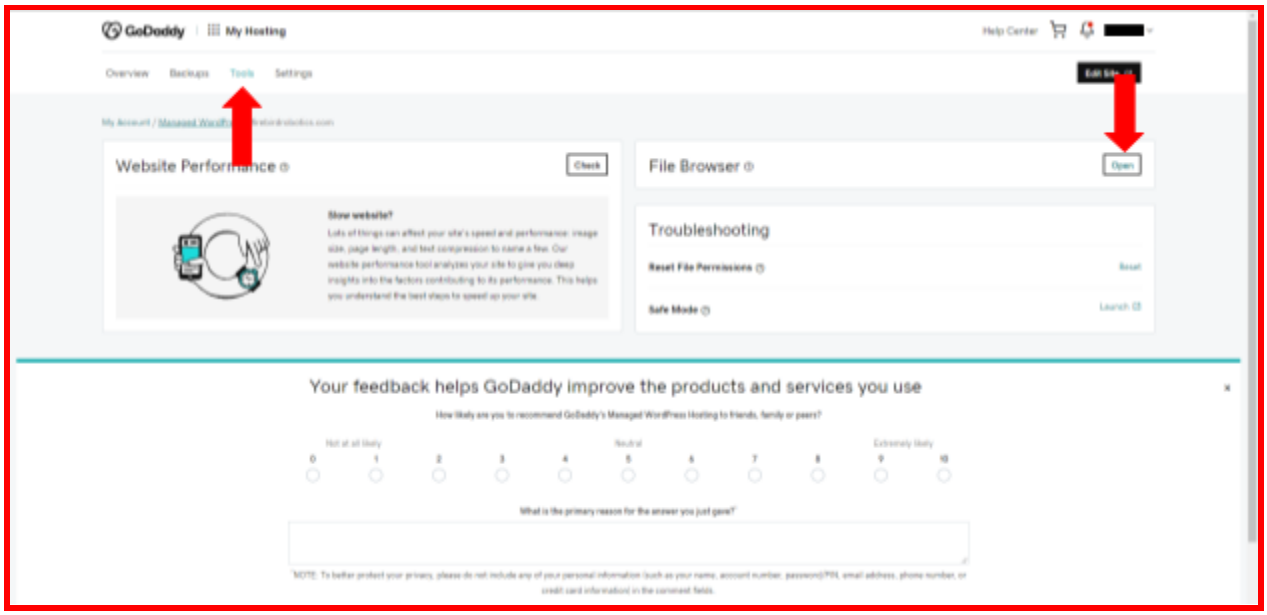
Putting the App on the Internet

If you have permission from whoever is in charge of the team website (firebirdrobotics.com), and you want to upload the changes you have made onto the live version of the scouting app, you will need to log in to the team's GoDaddy account (so go find whoever is in charge and ask them).

After logging in to the GoDaddy account, there are 2 places you need to go. First, you'll need to update the front-end of things. Under 'Managed WordPress', click on the button that says 'Manage' towards the right side.

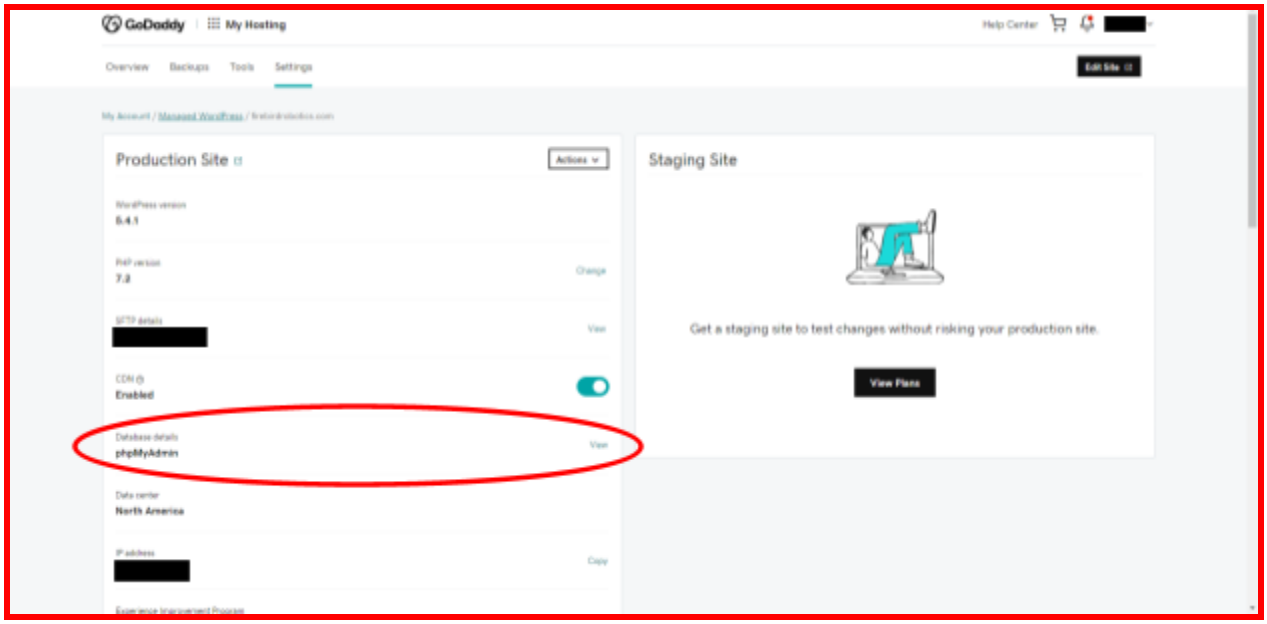


Afterwards, click on the 'Tools' tab at the top and then under 'File Browser', click on 'Open'.



In File Browser, simply upload all of your code files into the folder called ‘scouting’, and replace the files if needed.

The second place that you need to update is the database. To get to PHPMyAdmin on GoDaddy, go to the ‘Settings’ tab. Under ‘Production Site’, there should be an area labeled ‘Database details’.



Simply follow the instructions to get to the database, and make whatever changes you need to make.

Ending Note & Possible Improvements for the App

I developed the scouting app in my freshman year (2018) with a certain ***Mr. Jones***. This mentor had decades of computer programming experience, with a specialty in web programming, so you could imagine that it was not especially difficult for me to get help while making this project. However, that may not be the case for you, the reader, an assumed inexperienced programmer that does not have the guidance of a Mr. Jones to walk you through every single thing day in and day out. So, this guide was made to try and help you through the process of editing the scouting app, and perhaps act a little bit like a Mr. Jones for you.

While you are editing the scouting app, you might notice some sort of HTML or CSS syntax or something that is outdated, not proper, or has some other flaw. Please, by all means, feel free to change and improve these flaws, and keep in mind that I essentially made this app as a noob programmer freshman over a period of one school year and then never touched it again (until making this guide). As of now, I don't intend to make any major changes or improvements to the app, since I think that job can be left to many other people who have the motivation to do so. With that in mind, the following is a list of improvements or changes to the app that have been suggested by many different people to me over a period of multiple years. If you want to help join in on this project but don't know what to do, these are all ideas that you should feel free to try and implement into the scouting app. (Note: these ideas are also listed on the GitHub page for the scouting app, but these are just more detailed descriptions of them).

As a final note, don't worry if you feel like you have no idea what you are doing; that's the life of a programmer, and the only way to get to know what you're doing is by constantly making mistakes and experimenting. In fact, I had to basically re-learn most of the stuff in the scouting app when writing this guide, so my apologies if I missed something or an explanation seems incorrect (lol). Again, remember that the internet is your best friend when programming, so don't be afraid to look things up.

Account Recovery/Remind Users of their Username & Password

On almost all sites that have a login (in fact I can't think of one that doesn't do this), there is a way to recover your account if you've forgotten your login. There are two ways that I can think of doing this — emailing users their current username and password (which might be kind of risky if you accidentally email someone else's login, so I don't think I would take this approach), or emailing users a link to reset their password.

Both of these methods make use of email, but luckily for you, freshman me thought it was a great idea to just randomly ask people for their emails in the sign up, so you don't have to bother with getting peoples' emails. Instead, this would mainly focus on using some back-end code to find a user's email based on their entered username (in a new 'recover account' page) and then email them a safe link so that they can reset their password. As always, remember that the internet is your friend and there are tons of resources on YouTube and other sites like stackoverflow to help.

Automatically Importing Teams for Competitions from thebluealliance.com

thebluealliance.com is a commonly known website in FRC, and it shows all of the teams that participated as well as the match results for all competitions worldwide. They have open-source code, so it likely wouldn't

be too difficult to use some HTML link tags to get their data on what teams are participating. The best way I can think to do this is to make a new page in the 'Admin ONLY' section of the scouting app to select the competition and import those teams into the 'teams' table.

This project would probably take lots of googling and more, since this isn't exactly something I think would be common on most websites. One resource I would recommend for doing this is finding someone on the FRC Discord server that has experience with doing this kind of stuff.

Data Analysis of the Collected Data

One thing that would be really helpful and improve the scouting app's function would be to add data analysis for the data collected from Stand Scouting. I'm not entirely sure how to do this, since I'm not sure what the person who suggested this exactly had in mind, but it sounds like a significantly important improvement if you can get it done.

There are many ways that you could do this project, but what I would suggest is finding someone else's completed data analysis tool and just throwing it into a new page on the scouting app (rather than making your own data analysis stuff). Since data analysis is really only for numerical data, you would just take the fields from the 'robots' table that are restricted to numbers only and then have them as options to compare in whatever data analysis tool you implement (I'm envisioning just some sort of graph tool). Again, the FRC Discord and the internet are all good resources for doing this project.

Allowing Other Teams to Use the Scouting App

This project is probably the most difficult out of all of the mentioned improvements, since it would require the most change to the scouting app. The person who suggested this improvement intended for each team to have their own individual data table, rather than a shared data table between all of the users (which makes sense, since if two users are scouting the same robot during the same match, one user's data gets replaced).

This requires the users to be sorted in a different, more complex way than they are now, based on what team they are a part of. This could be done a number of ways, but the best way I could think of doing this is having the user input their team number in place of the 'Signup Code' field when they register for the scouting app.

The main issue of this project is actually sorting ***the data*** based on who collected it. As the scouting app works now, the data is all just under a single table with data being categorized with 2 fields, the team number and round number (if there is another entry with these same 2 fields, it will replace the one already there). By adding another field to categorize the data (user's team number), you could potentially have a way to show only the data submitted from one team.

In all honesty, I don't recommend trying to do this project, as there is just a lot that needs to happen for this to take place, and I'm sure there are more professional programmers in FRC that have already done something similar with a scouting app of their own. I think that just keeping the scouting app as a project for our team can be a good learning tool for newer programmers. Although, if you reeeeaaaaally want to try this project, then go for it (it's not like I can stop you lol).