Criteria C: Development

The product was developed according to the designs created.[1] following techniques were used in the creation of a website for managing students under the guidance of the swimming department.

1. Use of cascading style sheets (CSS3).
2. Use of Google Fonts.
3. Use of HTML5.
4. Use of JavaScript.
5. Use of jQuery.
6. Use of AJAX.
7. Use of Bootstrap framework.
8. Use of PHP to interact with database and web server.
9. Database creation using MySql and PhpMyAdmin.
10. Use of Photoshop to create UI samples.
11. Use of CanvasJS to create charts.
12. Use of a virtual Local host server (XAMPP)
13. Use of PHPmyadmin

[1] Criterion B

Graphical user interface[2]

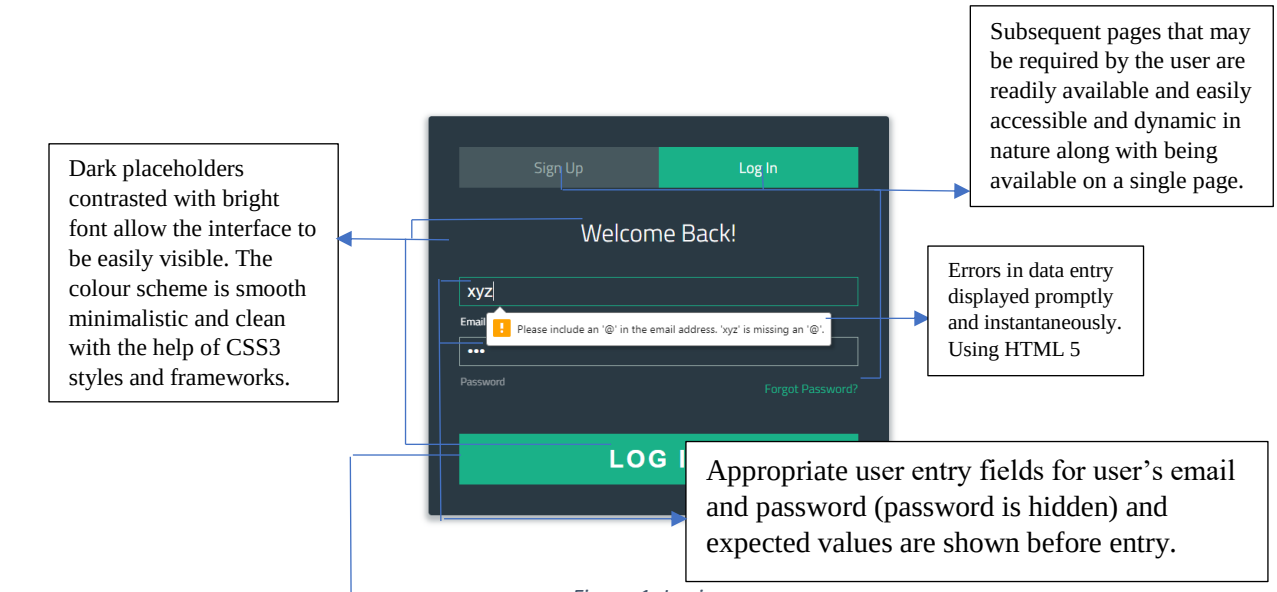


Figure 1: Login page.

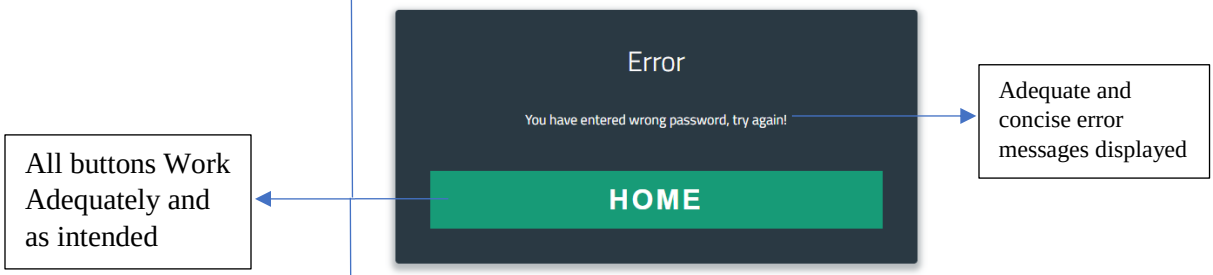


Figure 2: Error page.

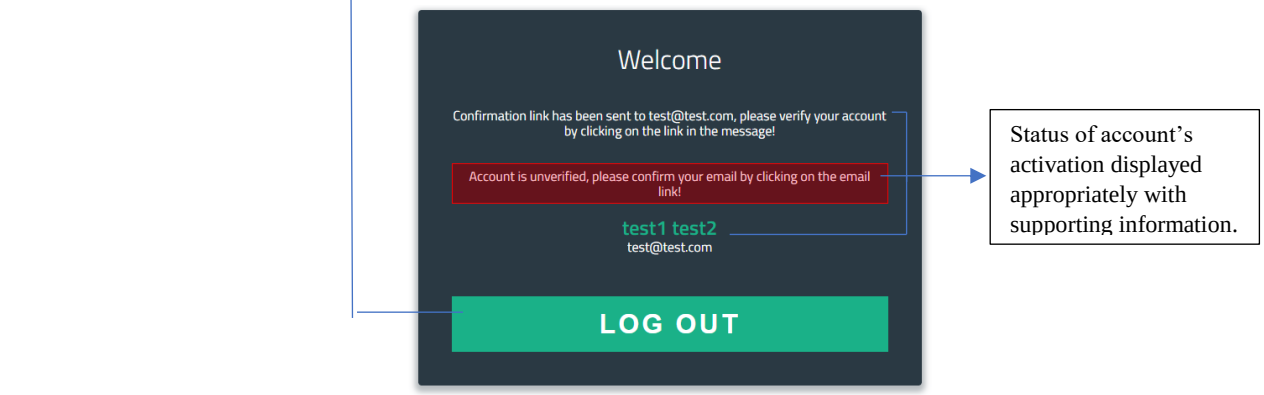


Figure 3: Welcome/ login page

[2] Refer to Criteria A: Planning: Success Criteria: "*x*" (10)

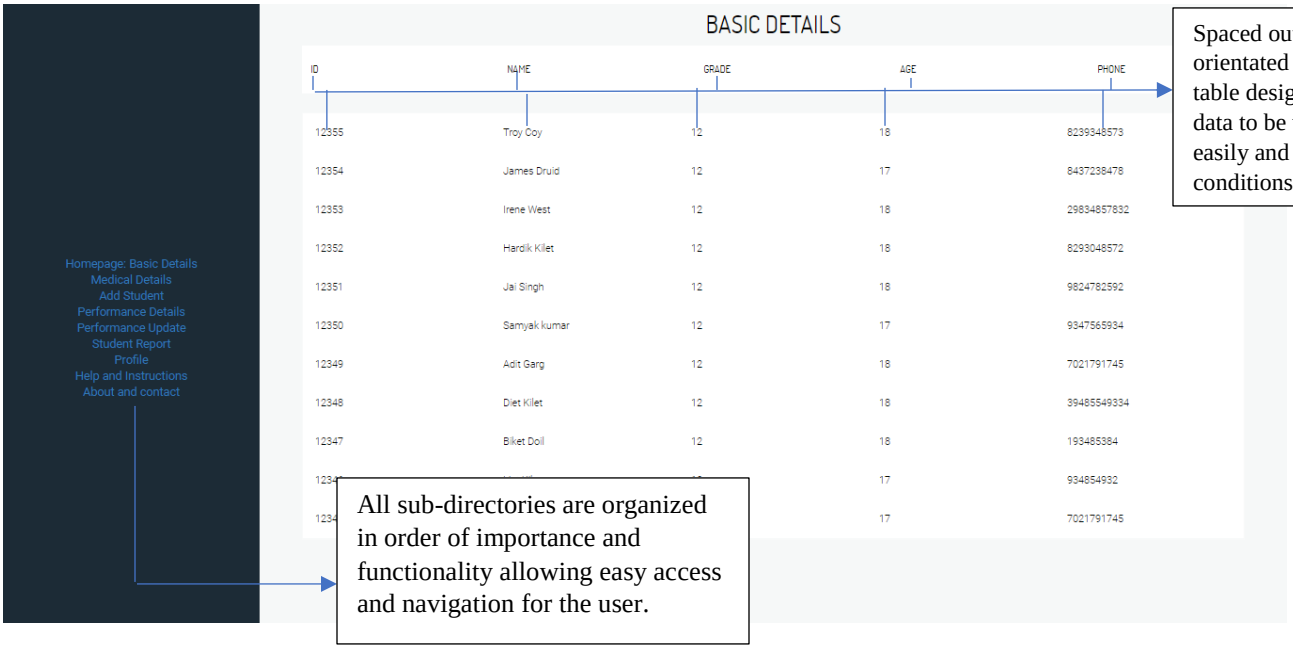


Figure 4: General Detail Pages (Medical and basic details)

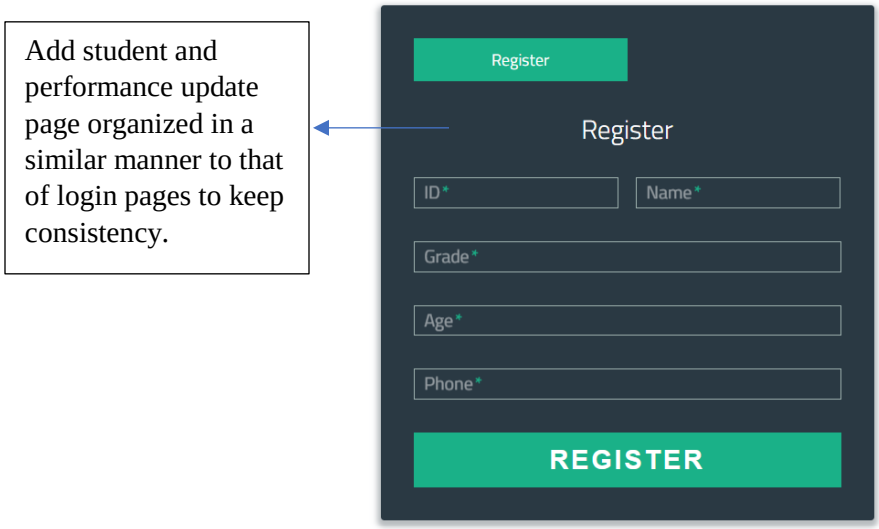


Figure 5: Data entry pages (Add Student and Performance Update page)

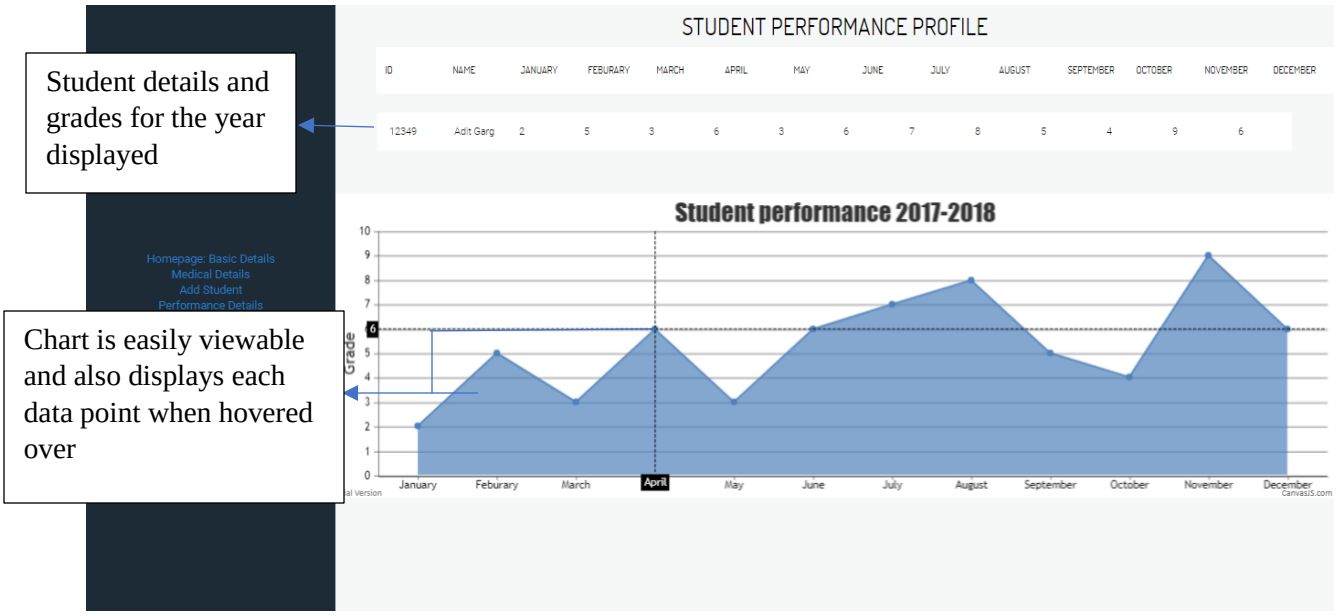


Figure 6: Student Performance profile.

Methods and Techniques Used

The techniques and features shown above was made functional using an integration of HTML5 and CSS3, which was enhanced by JavaScript and jQuery to provide a user-friendly experience. This website also employed Bootstrap framework to use its enhanced classes to build a well oriented user interface efficiently which also allowed the website to be responsive, by being able to scale equivocally on different screen sizes, but, to some extent but was limited due to the form of data recorded, this framework was further modified and overwritten by custom CSS3 and HTML5 scripts to provide a tailored experience for the user.

However, it should be noted that the client would be using the laptop provided by the school mostly to use this application.

2.01 Login system

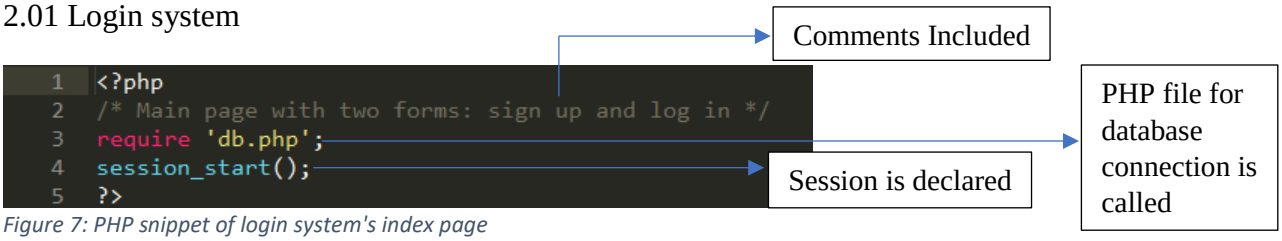


Figure 7: PHP snippet of login system's index page

As seen in the snippet above, at beginning of each page a PHP script is run to connect the front end with the back end and then a session is started to declare global/ session variables allowing for the data to be to some extent dynamic in nature. This allows the system to stay up to date as new users may register, login, or might not have verified their account, this increases the security of the system. Also, allowing the underlying code itself to stay well organized and easily understandable with the help of comments, so that in the future further developments could easily be made.

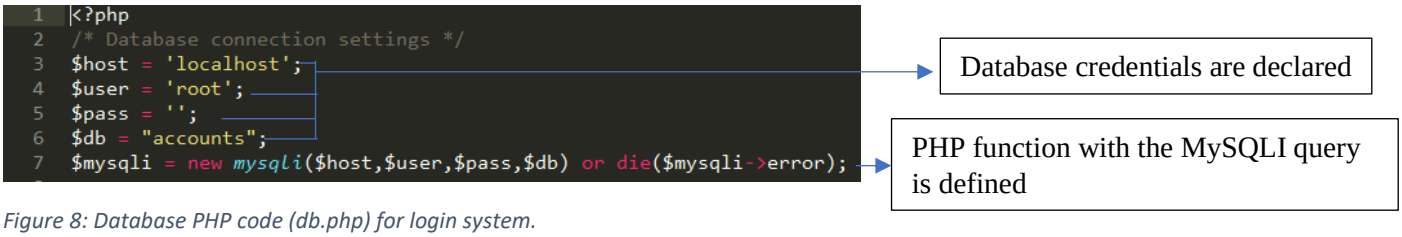


Figure 8: Database PHP code (db.php) for login system.

As seen in the code above, I have employed MySQLI instead of MySQL because MYSQL functions are phased out in later versions of PHP 7. When this function is run and if it's unsuccessful in connecting to the database it will throw an error.

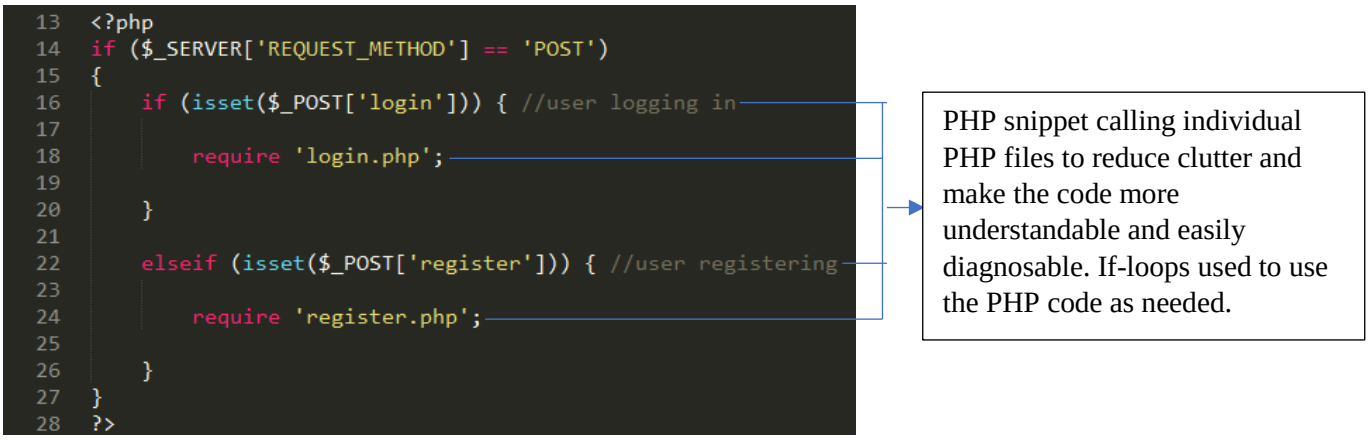


Figure 9: PHP snippet for calling different PHP files for each function for the index page of the login system.

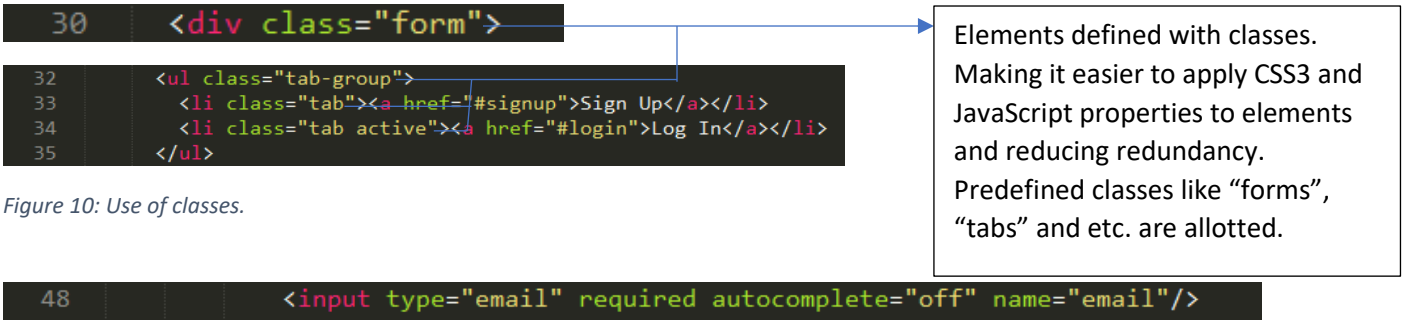


Figure 10: Use of classes.

Figure 11: HTML5 elements deployed

As seen in the code snippet above certain HTML5 elements, namely: Type and name, required, and autocomplete property are employed to make the interface user friendly and with the use of these elements it made the development cycle much more efficient. Type "email" and name "email" allow to run auto checks and the rest defines the constraints of data entry.

Figure 12: Individual Libraries being called.

As seen above individual libraries and resources are being called as scripts.

```php
<?php
if( isset($_SESSION['message']) AND !empty($_SESSION['message']) ):
    echo $_SESSION['message'];
else:
    header( "location: index.php" );
endif;
```

Figure 13: Error File

The following error file is called in underlying functions/ files for the system ("verify.php", "login.php", and etc. whenever the system runs into an error the PHP code will redirect user to the error page or return the user to the index page. With the use of if-loop the system is able to have separate outcomes for each scenario and for error a message is sent and the session variable is declared so that message can be retrieved from any page that has a session running. Echo is used to provide error messages in the console in case the program fails to specify the error or if it glitches out (which never happened during testing).

```css
*, *:before, *:after {
  box-sizing: border-box;
}
```

"*" is used to apply CSS3 styles to the entirety of the page and "," is used to apply styles to multiple identifiers at the same time and "*: before" allows to apply styles to a certain child node. (before can be replaced with a certain identifier and can be used as conditional formatting and styling.

Figure 14: Styling: border box.

Here, Box sizing is defined as border-box this essentially removes any browser defined margins or padding around the webpage. Allowing me to apply styles and orientate pages based on the entirety of the browser and eliminates the requirement of multiple browser conditional styling done via web kits. However, it should be noted that due to Internet Explorer being phased out from most of school's computers and the primary browser being "google chrome" the rest of the styling will be operational and would not require additional tweaking.

```html
<div id="navbar1">
    <ul id=Nvel>
      <li><a href="index.php">Homepage: Basic Details</a></li>
      <li><a href="medicaldetails.php">Medical Details</a></li>
      <li><a href="register.php">Add Student</a></li>
      <li><a href="perfedetails.php">Performance Details</a></li>
      <li><a href="gradeupsite.php">Performance Update</a></li>
      <li><a href="st chart.php">Student Report</a></li>
      <li><a href="profile.php">Profile</a></li>
      <li><a href="instructions.html">Help and Instructions</a></li>
      <li><a href="about.php">About and contact</a></li>
    </ul>
</div>


  <div id="container">
<section>
  <h1>Basic Details</h1>
  <div class="tbl-header">
    <table cellpadding="0" cellspacing="0" border="0">
      <thead>
        <tr>
          <th class="weirdind">ID</th>
          <th class="weirdind">NAME</th>
          <th class="weirdind">GRADE</th>
          <th class="weirdind">AGE</th>
          <th class="weirdind">PHONE</th>
        </tr>
      </thead>
    </table>
  </div>
```

Id "navbar1", "container", "Nvel" used to employ bootstrap functions and styles to aid in creating navigational bars.

Anchor tags used to hyperlink other pages or subsections on the page.

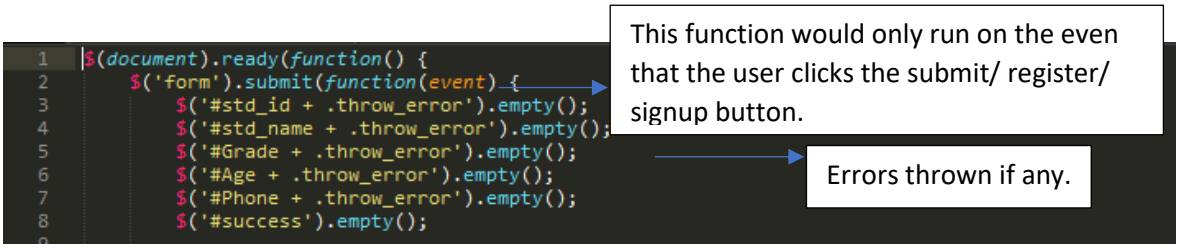
Tables created and browser defined styling removed to apply custom CSS3 styling.

Figure 15: Dynamic charting code.

As seen in the code snippet above, the code above calls CanvasJS libraries. The code above calls for values from the database accordingly and defines them as datapoints.

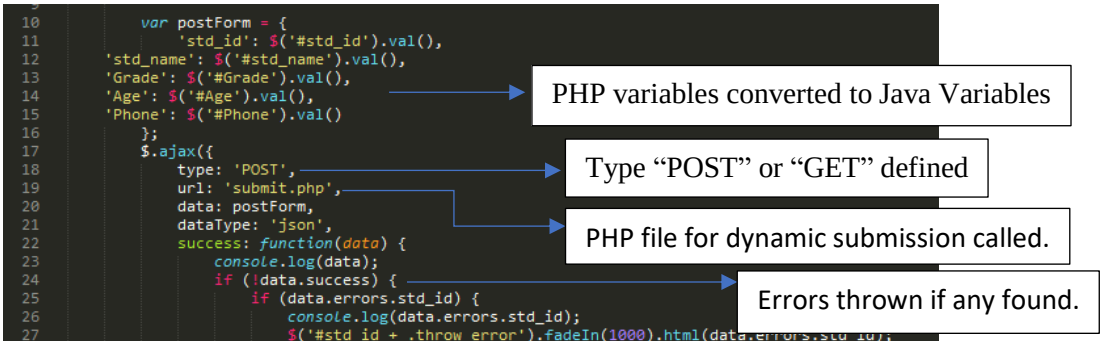
Figure 16: JSON script for dynamic CanvasJS charts.

The code snippet above defines constants for charts and defines visual features associated with the chart.




```
$(document).ready(function() {
    $('form').submit(function(event) {
        $('#std_id + .throw_error').empty();
        $('#std_name + .throw_error').empty();
        $('#Grade + .throw_error').empty();
        $('#Age + .throw_error').empty();
        $('#Phone + .throw_error').empty();
        $('#success').empty();
```

Figure 17: PHP variables retrieved from the form submission




```
        var postForm = {
            'std_id': $('#std_id').val(),
    'std_name': $('#std_name').val(),
    'Grade': $('#Grade').val(),
    'Age': $('#Age').val(),
    'Phone': $('#Phone').val()
        };
        $.ajax({
            type: 'POST',
            url: 'submit.php',
            data: postForm,
            dataType: 'json',
            success: function(data) {
                console.log(data);
                if (!data.success) {
                    if (data.errors.std_id) {
                        console.log(data.errors.std_id);
                        $('#std_id + .throw_error').fadeIn(1000).html(data.errors.std_id);
                    }
                    if (data.errors.std_name) {
                        console.log(data.errors.std_name);
                        $('#std_name + .throw_error').fadeIn(1000).html(data.errors.std_name);
                    }
                    if (data.errors.Grade) {
                        console.log(data.errors.Grade);
                        $('#Grade + .throw_error').fadeIn(1000).html(data.errors.Grade);
                    }
                    if (data.errors.Age) {
                        console.log(data.errors.Age);
                        $('#comment + .throw_error').fadeIn(1000).html(data.errors.Age);
                    }
                    if (data.errors.Phone) {
                        console.log(data.errors.Phone);
                        $('#comment + .throw_error').fadeIn(1000).html(data.errors.Phone);
                    }

                } else {
                    checkdatabase();
                    $('#success').fadeIn(1000).append('<p>' + data.posted + '</p>');
                }
            }
        });
        event.preventDefault();
    });
```

Figure 18: Dynamic submission of data (using AJAX, JSON, PHP) to the database and displaying errors if any.

```
    function checkdatabase() {
        $('#output').empty();
        $.ajax({
            url: 'data.php',
            dataType: 'json',
            cache: false,
            success: function(data) {
                var i = 0;
                $.each(data, function(index) {
                    i++;
                    console.log(data[index]);
                    $('#output').append("<tr>");
                    $('#output').append("<td>" +  data[index].std_id +"</td>");
                    $('#output').append("<td>" +  data[index].std_name +"</td>");
                    $('#output').append("<td>" +  data[index].Grade +"</td>");
                    $('#output').append("<td>" +  data[index].Age +"</td>");
                    $('#output').append("<td>" +  data[index].Phone +"</td>");
                    $('#output').append("<td>" +  data[index].perf_grade +"</td>");
                    $('#output').append("</tr>");
                });
            }
        });
    }
    checkdatabase();
});
```

Figure 19: Dynamic retrieval of data from database and defining it as a function

The AJAX code's snippet above, refers to the id tag "output" on the relevant front-end PHP page. The browser calls the id "output" and upon the call the data in the tag is replaced with variable data gathered dynamically from the database and predefined constants until all data from the table is gathered.

```
<div class="tbl-content " >
    <table cellpadding="0" cellspacing="0" border="0">
      <tbody id="output">

      </tbody>
    </table>
  </div>
```

Figure 20: PHP file's code with id tag "output"

Essentially, a table is constructed using a for-each loop which also eliminates the need for a counter to count rows in the database. This allots a table structure and inputs data gathered from a particular index in the database with the following code "data[index].std_id" which is enclosed with table elements.

SQL query

```
// Set session variables to be used on profile.php page
$_SESSION['std_id'] = (isset($_POST['std_id']));
$_SESSION['std_name'] = (isset($_POST['std_name']));
$_SESSION['Grade'] = (isset($_POST['Grade']));
$_SESSION['Age'] = (isset($_POST['Age']));
$_SESSION['Phone'] = (isset($_POST['Phone']));

// Escape all $_POST variables to protect against SQL injections
$std_id = $mysqli->escape_string(isset($_POST['std_id']));
$std_name = $mysqli->escape_string(isset($_POST['std_name']));
$Grade = $mysqli->escape_string(isset($_POST['Grade']));
$Age = $mysqli->escape_string(isset($_POST['Age']));
$Phone = $mysqli->escape_string(isset($_POST['Phone']));
$sql= "INSERT INTO basicdetails (std_id, std_name, Grade, Age, Phone)  VALUES ('$std_id','$std_name','$Grade','$Age', '$Phone')";
mysqli_query( $mysqli , $sql);
ob_end_flush();

$mysqli->close();
```

Figure 21: Session variables defined and then uploaded to the database after submission button is clicked on.

As seen in the code snippet above, all variables are “escaped” to increase security and prevent SQL injections, which are highly unlikely as the website will only exist on the school’s servers. However, this is done in case the school decides to develop the product further and make it available to other branches which would only be accessible through intranet, VPN, or other network frameworks.

Note: “isset” is employed to eliminate any “undefined variable” errors or numerous irrelevant errors by setting the variable.

```php
<?php
/* User login process, checks if user exists and password is correct */

// Escape email to protect against SQL injections
$email = $mysqli->escape_string($_POST['email']);
$result = $mysqli->query("SELECT * FROM users WHERE email='$email'"

if ( $result->num_rows == 0 ){ // User doesn't exist
    $_SESSION['message'] = "User with that email doesn't exist!";
    header("location: error.php");
}
else { // User exists
    $user = $result->fetch_assoc();

    if ( password_verify($_POST['password'], $user['password']) ) {

        $_SESSION['email'] = $user['email'];
        $_SESSION['first_name'] = $user['first_name'];
        $_SESSION['last_name'] = $user['last_name'];
        $_SESSION['active'] = $user['active'];

        // This is how we'll know the user is logged in
        $_SESSION['logged_in'] = true;

        header("location: http://localhost:8080/IA%20CS/");
    }
    else {
        $_SESSION['message'] = "You have entered wrong password, try again!";
        header("location: error.php");
    }
}
```

If loop to determine if any matching users already exist.

Figure 22: PHP code snippet verifying if the user exists on the database and the account is activated.

As seen in the snippet above, similar procedure is used during the registration process.

Database Setup

To test the database on my local machine, there were certain dependencies that had to be installed.

1. Install dependencies: PHP 7, MySQL, JSON, AJAX
2. To test the system on a virtual network, install XAMPP.
3. Put all relevant project files in the htdocs folder.
4. Launch XAMPP control panel, and start Apache and MySQL modules.
5. Change localhost port to 8080
6. Visit: http://localhost:8080/IA%20CS/login-system%20(1)/index.php

There were 2 databases created: tik and Accounts.

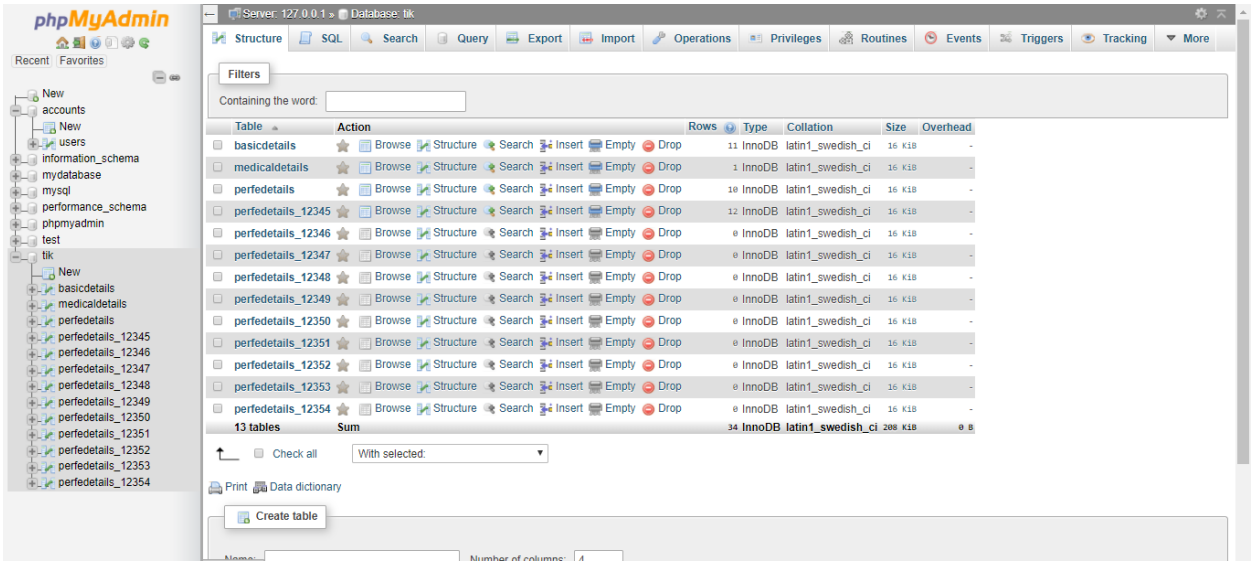


Figure 23: Database tik

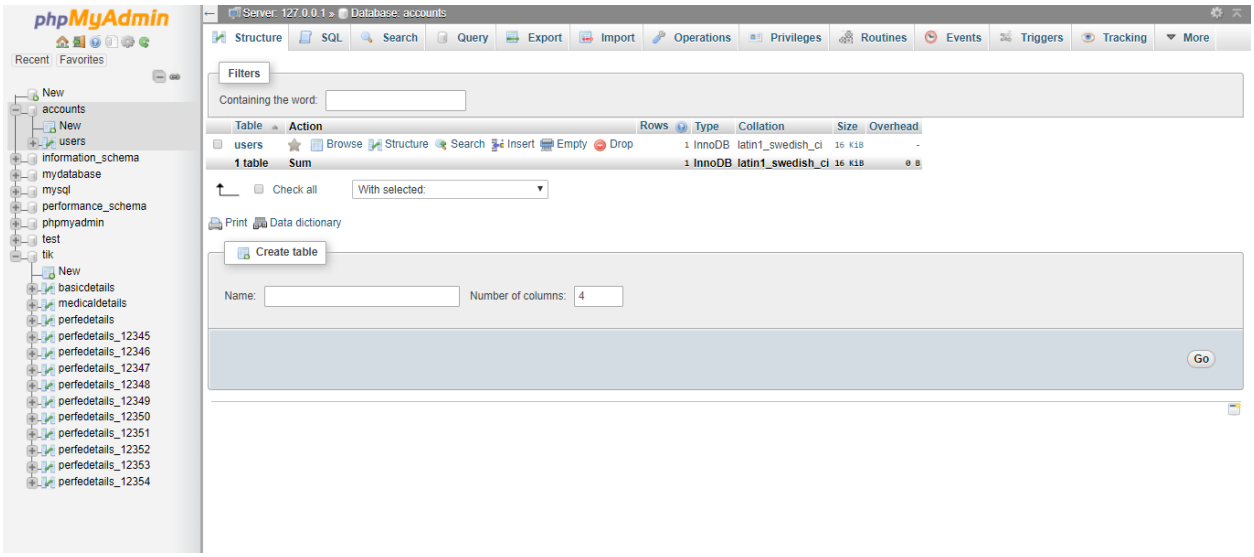


Figure 24: Database Accounts

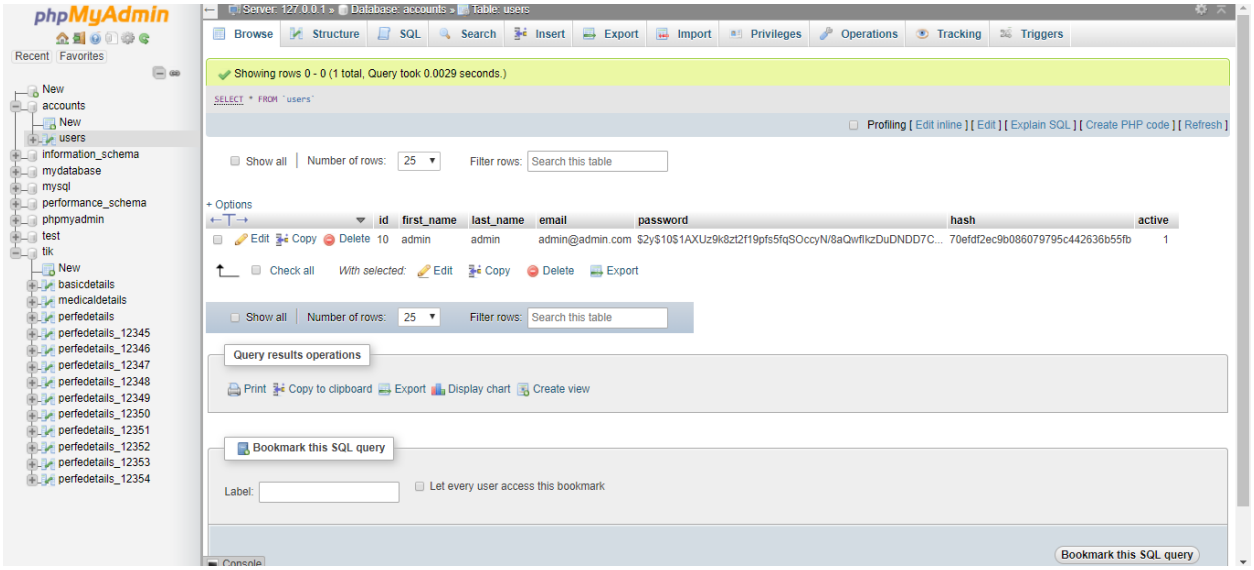


Figure 25: Database Accounts: users

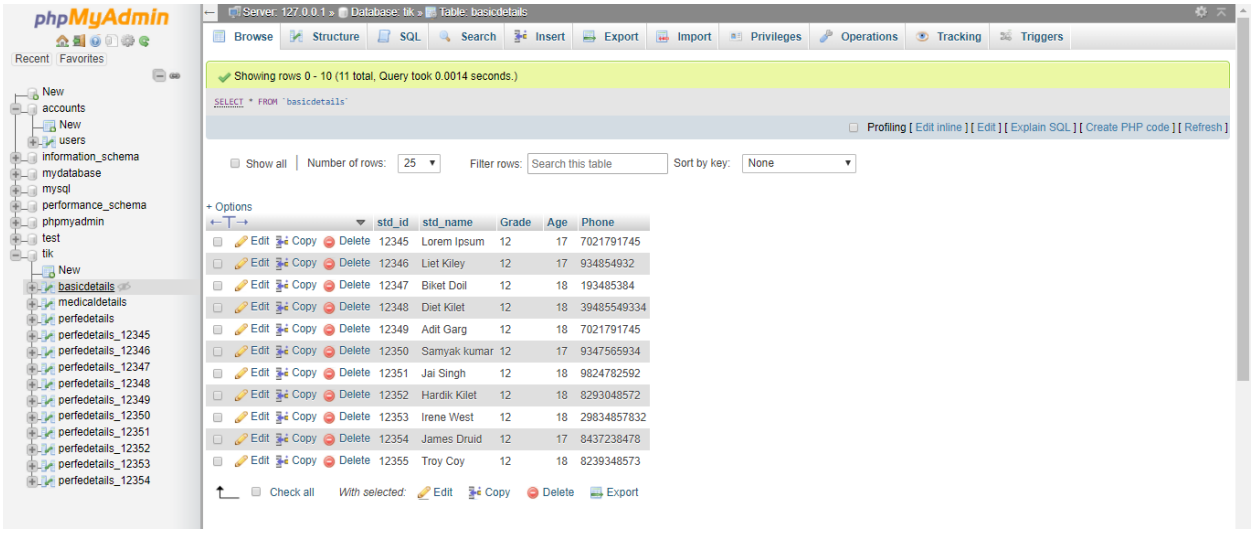


Figure 26: Database:tik : basicdetails

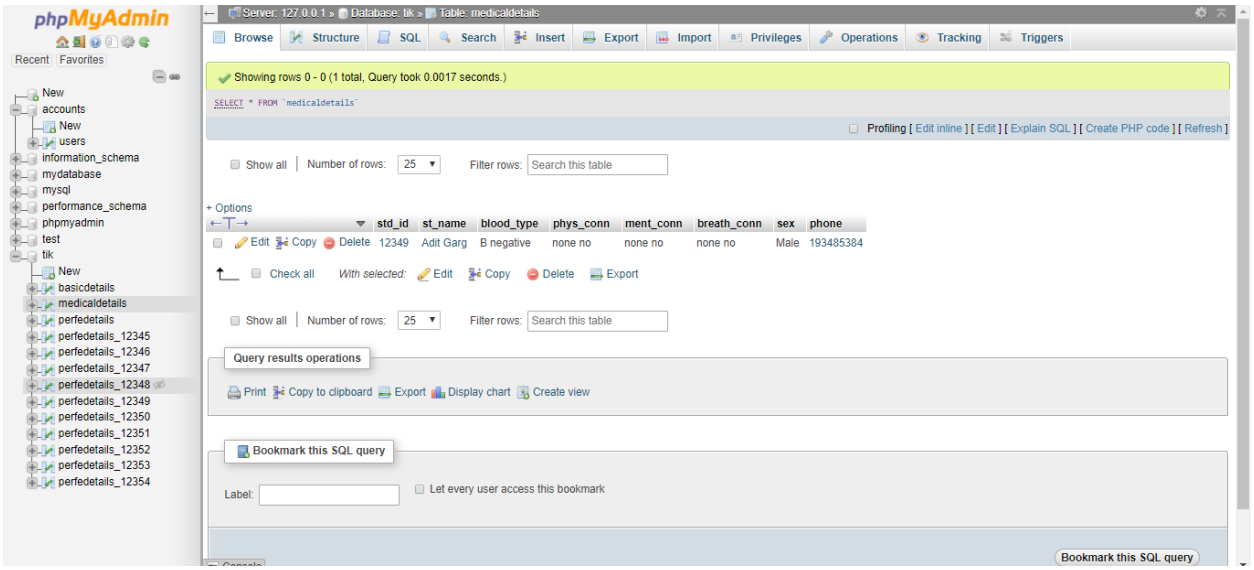


Figure 27: Database: tik: medicaldetails

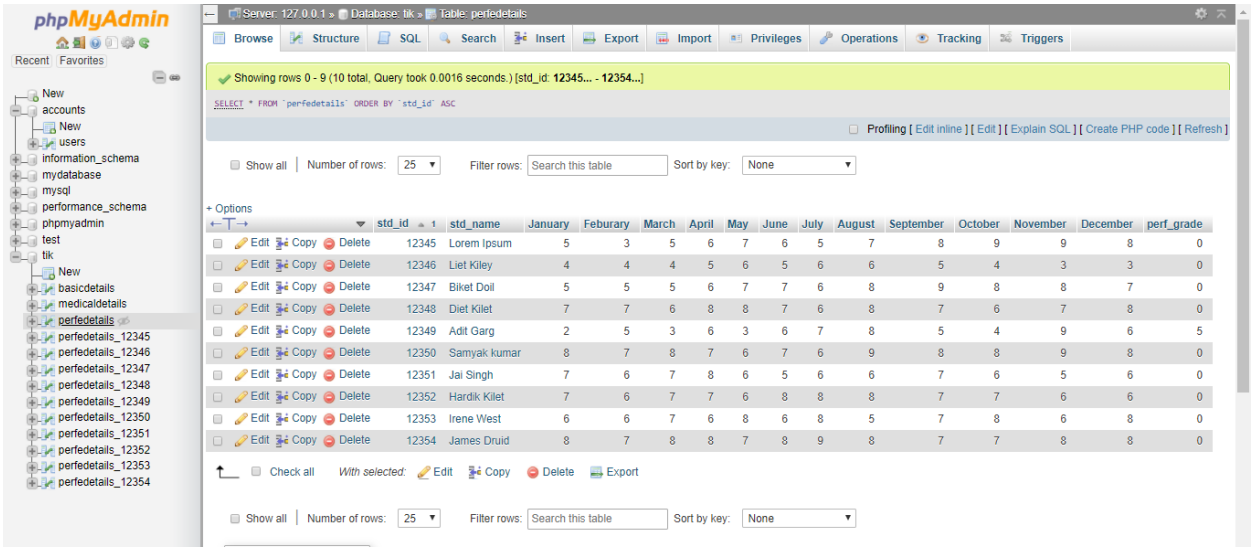


Figure 28: Database: tik: peredetails

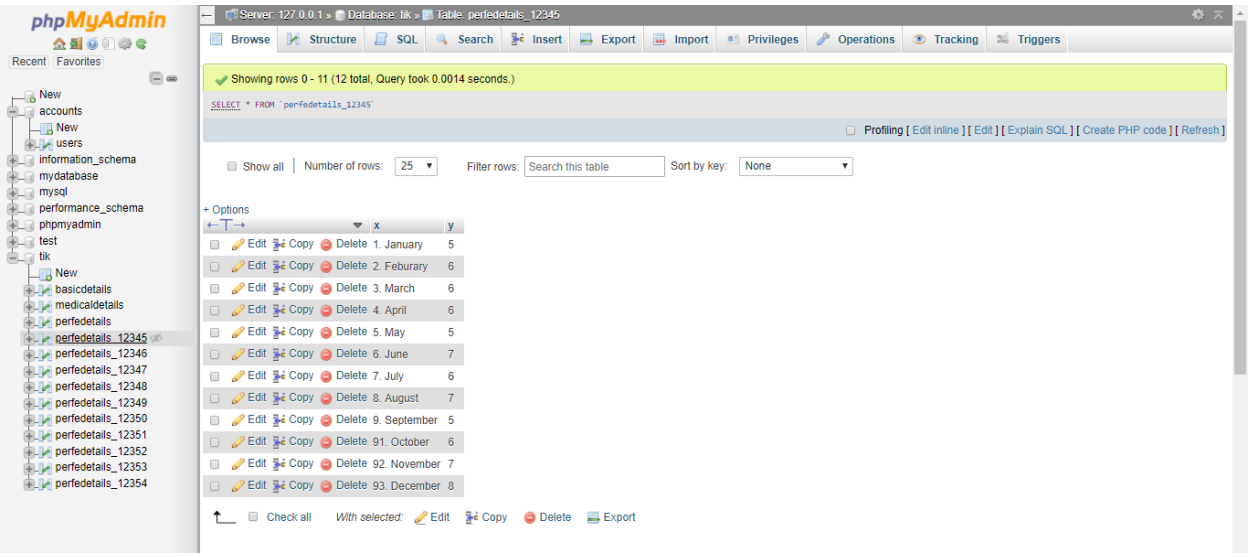


Figure 29: Database: tik: perfedetails_12345

To construct, individual student reports I had to create individual tables for each dummy-data student. And the rest perfedetails_12346 to 12354 have the same structure. Dynamic table creation for individual students could not be implemented in time, this feature could be further developed in the future under extensibility. This is also part of the reason only one student's report is generated.

Hosting

During the entirety of the development of the project, the project was developed using XAMPP's virtual hosting. However, later the project was uploaded to school's local servers.