Contents

[Appendix 81](#_Toc511632445)

[Evidence of investigation (observation, interview, questionnaire, documentation) 81](#_Toc511632446)

[Interview Questions 81](#_Toc511632447)

[First Interview with Client 1 (Alaap)-Beginner 81](#_Toc511632448)

[First Interview with Client 2 (Imran)-Intermediate 81](#_Toc511632449)

[First Interview with Client 3 (Xue)- Advanced 81](#_Toc511632450)

[Questionnaire 82](#_Toc511632451)

[Evidence of existing Current Systems 83](#_Toc511632452)

[Log 86](#_Toc511632453)

[Coding 87](#_Toc511632454)

[Class: DB 87](#_Toc511632455)

[Class: Exercise 88](#_Toc511632456)

[Class: Routine 90](#_Toc511632457)

[Class: Search 102](#_Toc511632458)

[Class: Timetable 106](#_Toc511632459)

[Class: User 111](#_Toc511632460)

[Autoloader 114](#_Toc511632461)

[Page: addowner.php 114](#_Toc511632462)

[Page: create.php 116](#_Toc511632463)

[Page: home.php 120](#_Toc511632464)

[Page: index.php 121](#_Toc511632465)

[Page: log.php 122](#_Toc511632466)

[Page: logout.php 124](#_Toc511632467)

[Page: profile.php 124](#_Toc511632468)

[Page: register.php 127](#_Toc511632469)

[Page: search.php 129](#_Toc511632470)

[Page: forgotpassword.php 131](#_Toc511632471)

[Page: timetable.php 134](#_Toc511632472)

## Appendix

## Evidence of investigation (observation, interview, questionnaire, documentation)

### Interview Questions

**Me:**

*a.) How do you plan and record your gym routines every week?*

*b.) How is each routine executed, give me an example if necessary?*

*c.) What is your biggest issue you face going to the gym?*

*d.) What feature would you want a new proposed system to have?*

### First Interview with Client 1 (Alaap)-Beginner

a.) When I started [few months ago] I would always follow my friends, so I did not have a structured weekly schedule on what I was going to do. Now however, I try aim to train the major muscle groups once a week.

b.) I do most of my workout in the school gym, and I will often be accompanied by more experienced peers. They teach me how to perform each exercise and we take turns to rest while the other does his rep. In a workout routine, I have a good idea of what I want to do, and try complete as many exercises I can within a set limit of time.

c.) I find it difficult to motivate myself to go to the gym on a constant basis. I also find it hard to do new routines because I don’t know how to do new exercises, and don’t want to get injured.

d.) I would like to be able to access a large variety of exercises and definitions on how to perform them.

### First Interview with Client 2 (Imran)-Intermediate

a.) I try train at least 3 times a week, I leave a day in-between workouts to let my body recover. My main priorities are arms, chests and abs. When I go to the gym I know which exercises I want to do. I do not have a plan, or recorded routines, I just do exercises I know how to do.

b.) When training arms, I always start with the most difficult exercise usually this is a barbell curl. As workout progresses I move onto lighter weight and finish with an ab workout.

c.) As workout progresses I tend to lose motivation due to fatigue. I also tend to do the same workouts repeatedly with no change to exercise variation. I also don’t know how to train legs because I was never taught by my friends.

d.) I think it would be good to have a weekly schedule, which I can plan, so my training plan is more structured and effective. Also, I would like to have access to knowledge on how to perform more exercises.

### First Interview with Client 3 (Xue)- Advanced

a.) Every week I have a set number of routines, I usually perform 5-6 days of weight lifting and 1 day of cardio or arest day. I have a dedicated day to arms, chest and back, shoulders and legs. I do have a plan, but I do not record it.

b.) Before every workout I start with a warm up, such as stretching. I then move onto my main exercises. I first do a light warm up set before moving onto working sets. For example, when I train chest, I always bench press first because that takes the most effort. After I will move onto exercises of lower weight such as cable fly. I often perform 3 sets of 10, and sometimes incorporate super sets or drop sets.

c.) My biggest issue is overtraining, sometimes I get injured due to the weight being too heavy or my technique being poor due to fatigue.

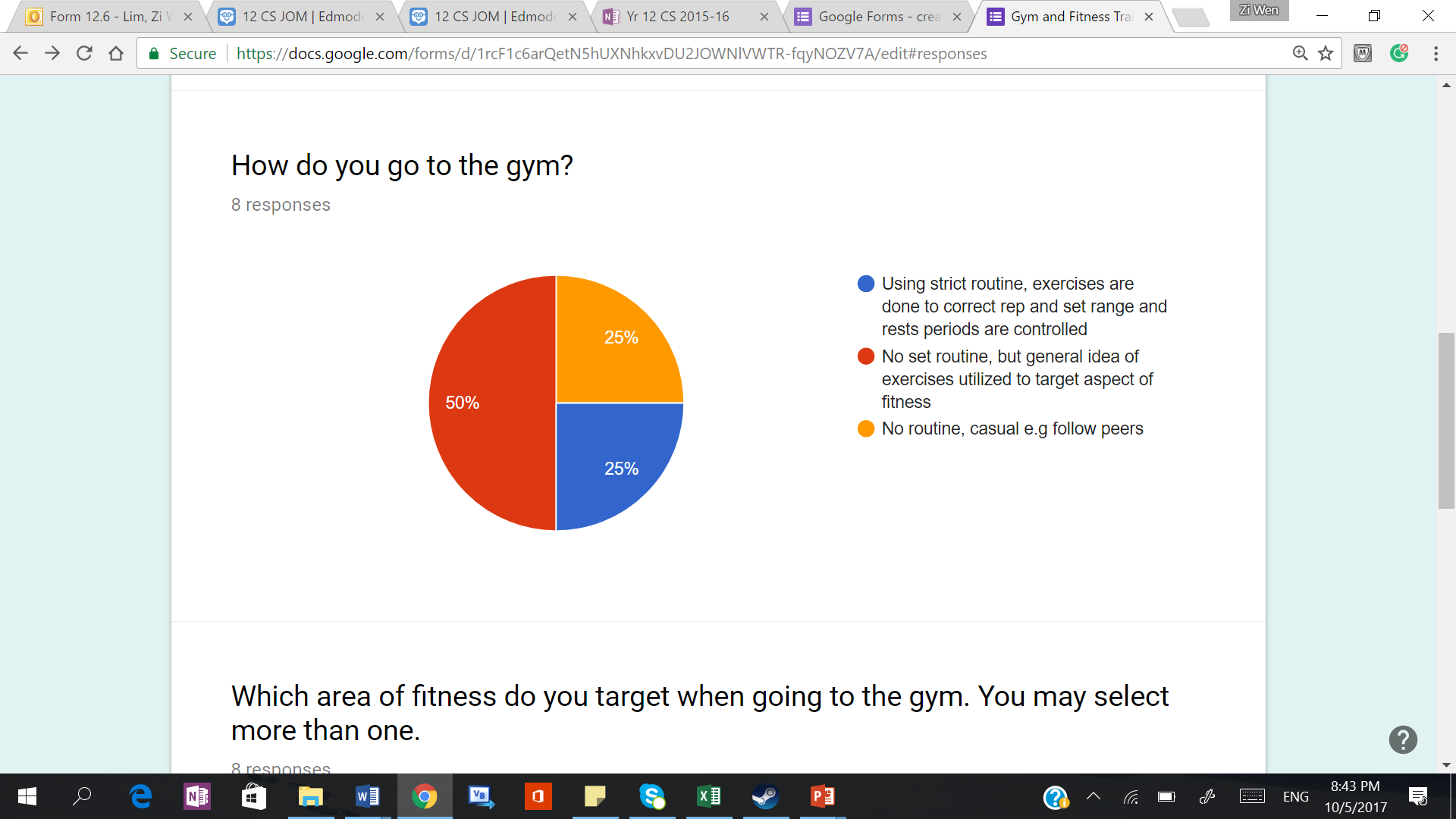
d.) It would be nice to use a dynamic training plan which can maximise my training efficiency, one that can update my intensity based on my previous feedback.

## Questionnaire

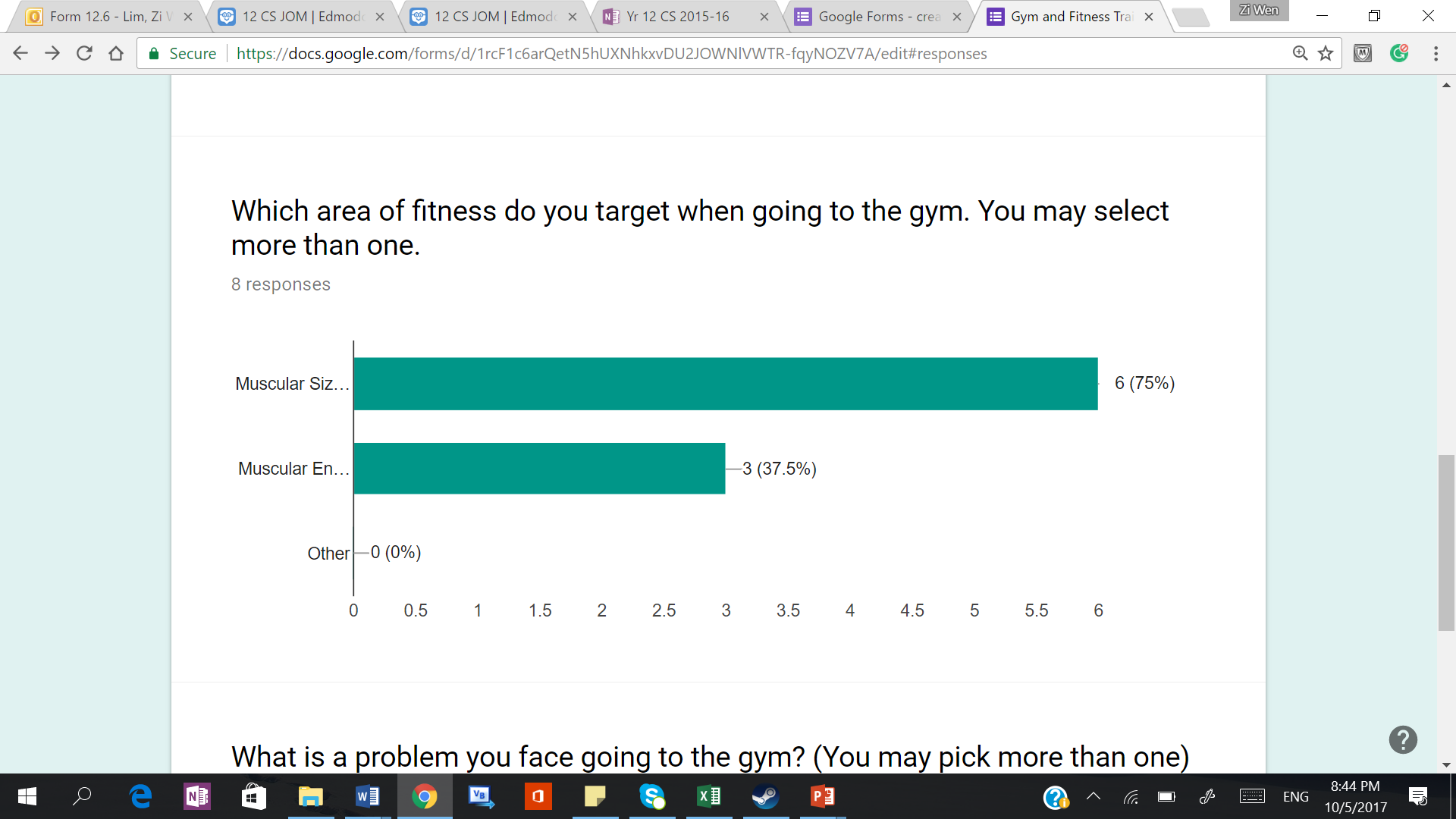
|  |
| --- |
| Fig 2.1.1 |



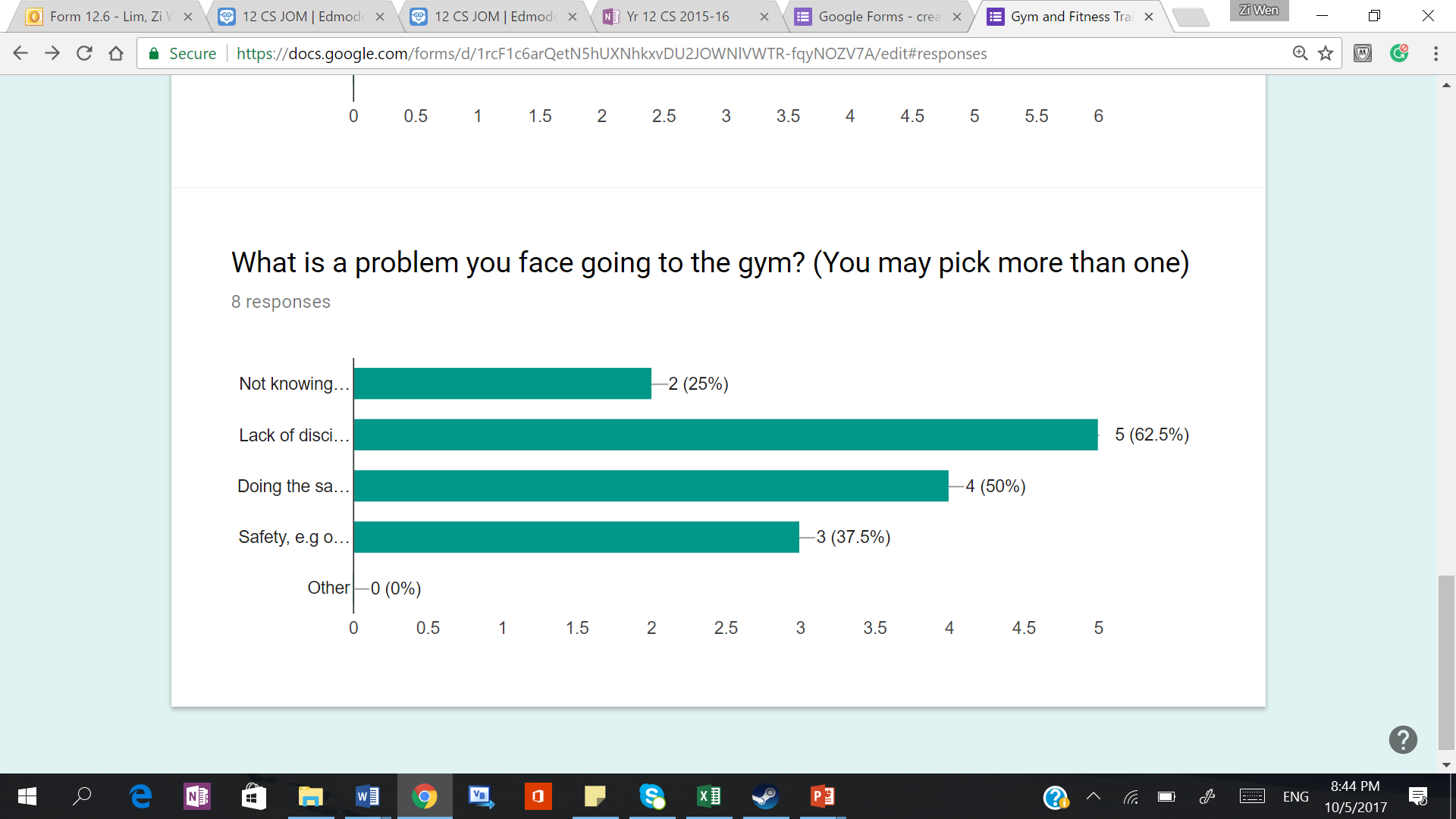
|  |
| --- |
| Fig 2.1.2 |



|  |
| --- |
| Fig 2.1.3 |



|  |
| --- |
| Fig 2.1.4 |



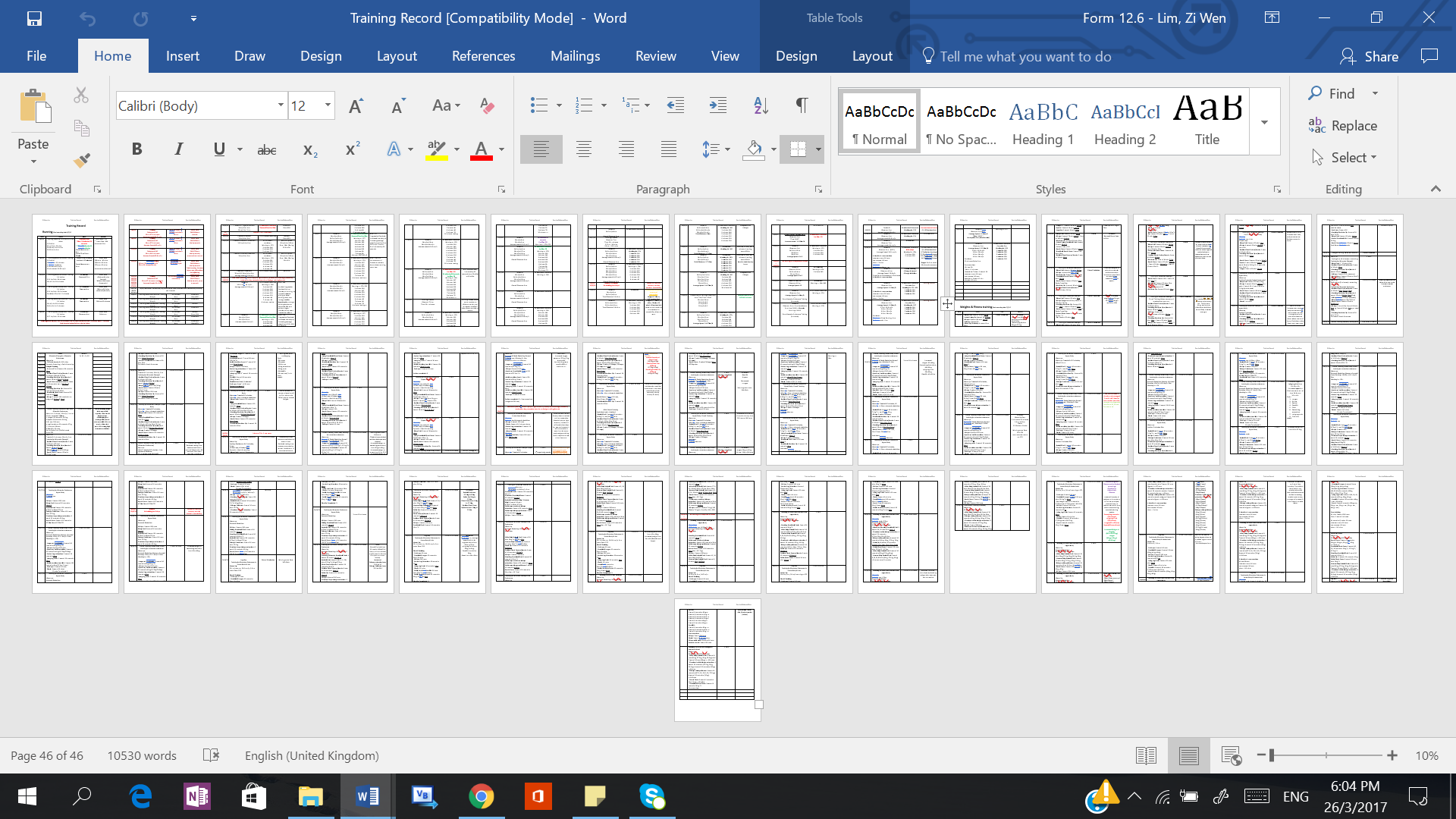
|  |
| --- |
| 1. Not knowing what to do and what routine to follow      1. Lack of discipline and motivation e.g not going on a constant basis      1. Doing the same workout repeatedly, with no change e.g to intensity or exercises      1. Safety, e.g over training / injury / not knowing how to perform exercise |

## Evidence of existing Current Systems

|  |
| --- |
| Fig 2.2.1 |

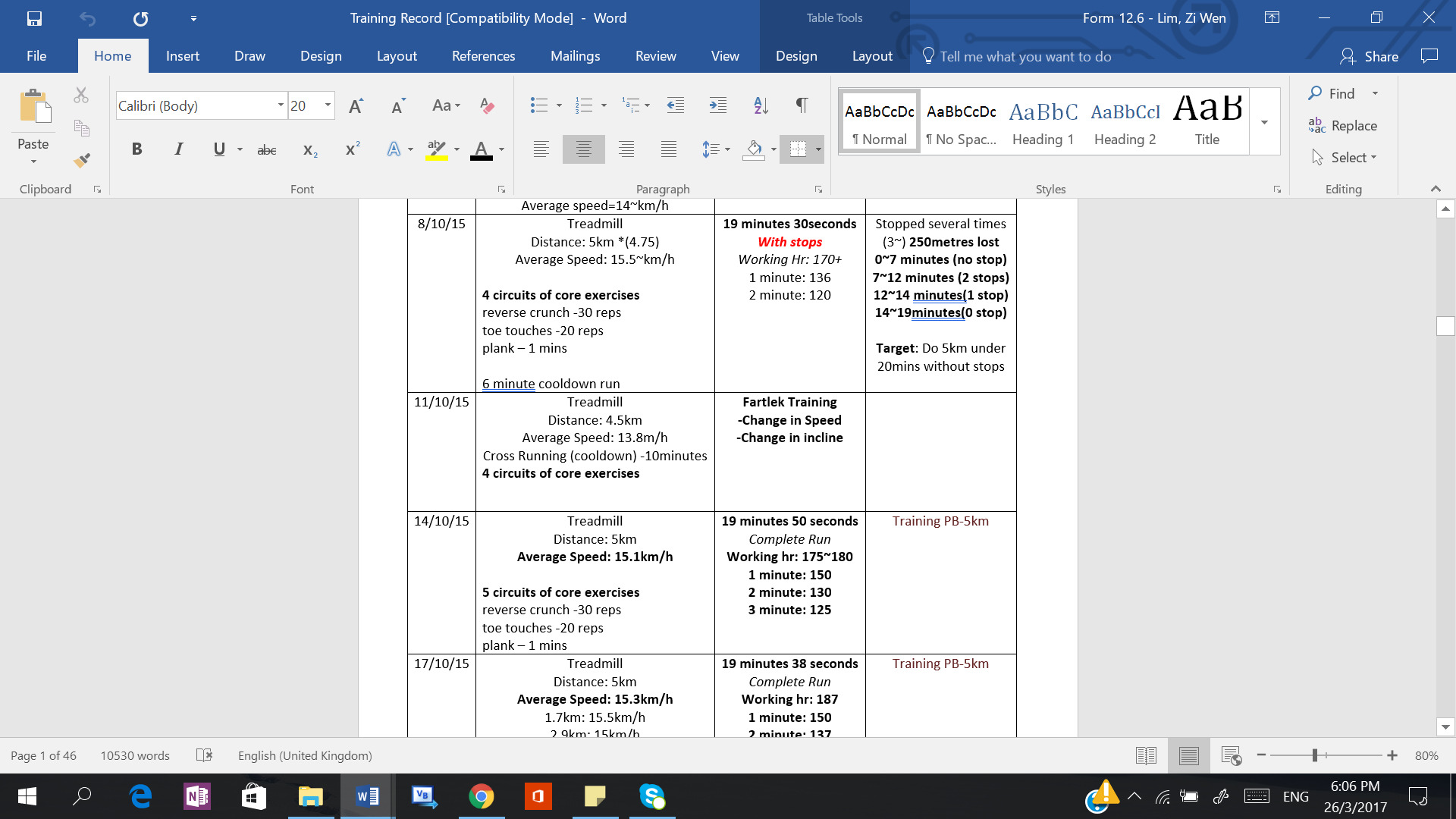
**Example of my workout plan (8 months from Feb 2015 to Oct 2015)**

*Split into running and gym training*

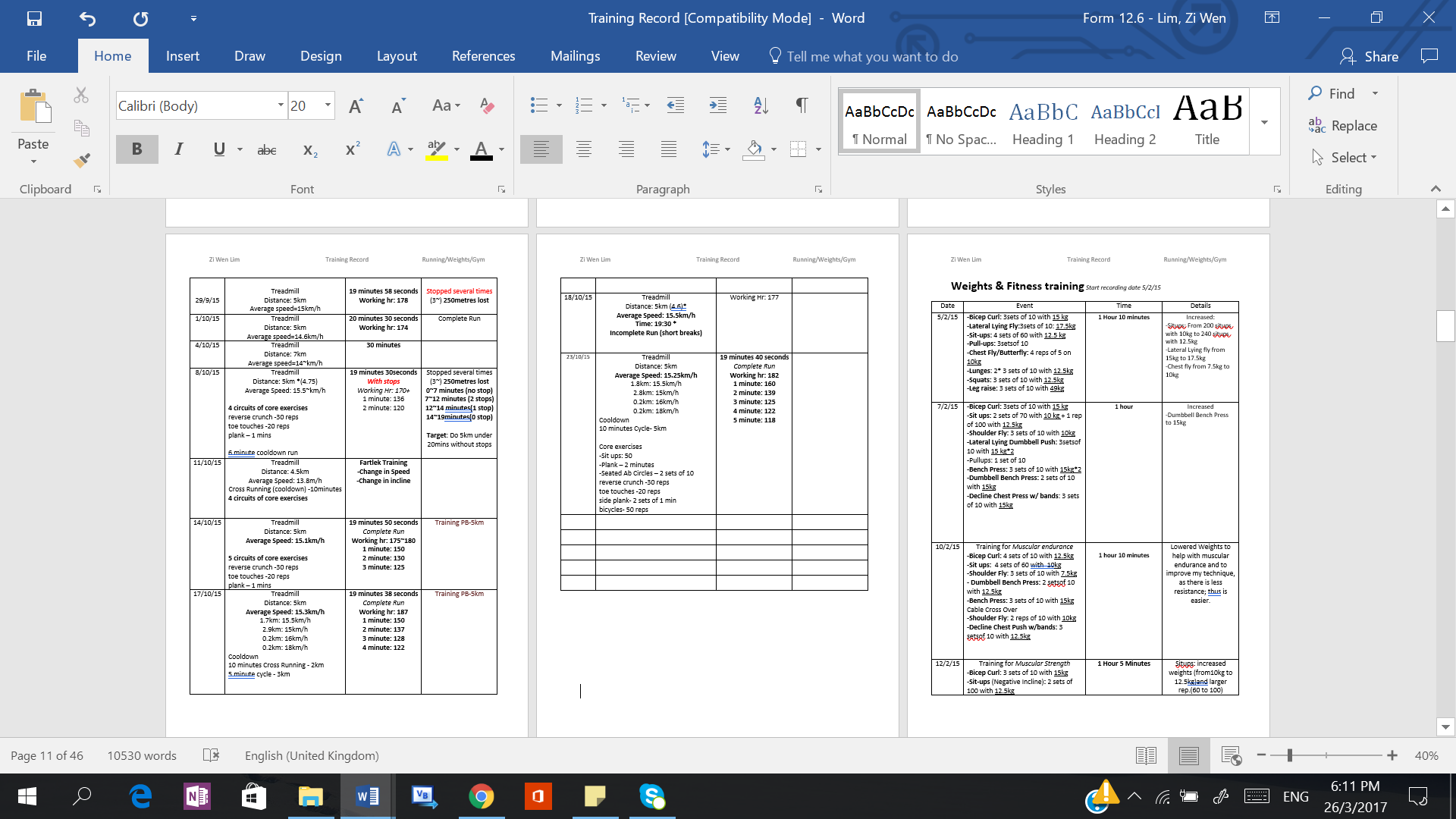
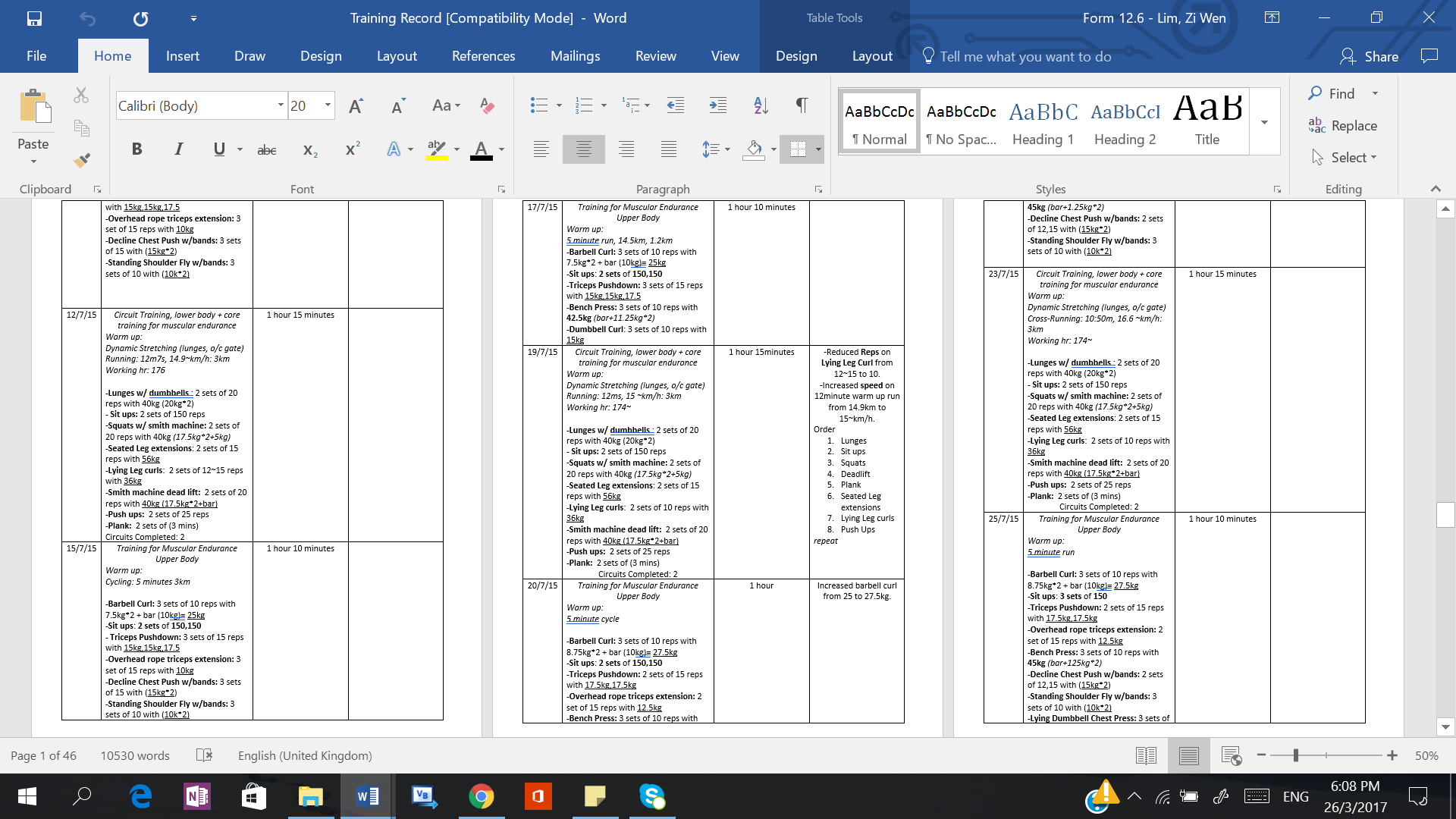


Extract of gym record (Running)

|  |
| --- |
| Fig 2.2.2 |

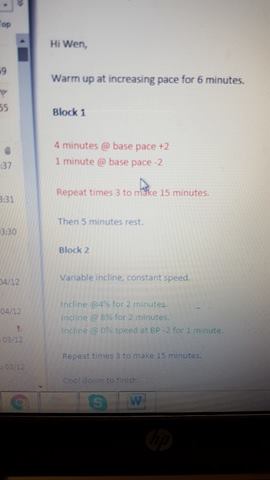


Extract of Gym Record (Weight Lifting)



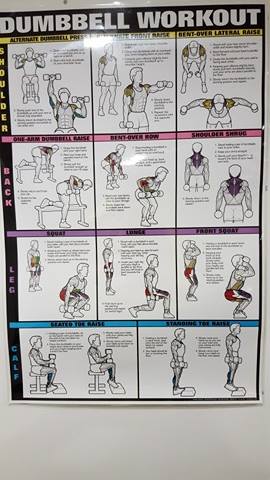
|  |
| --- |
| Fig 2.2.3 |

**Example of routine given by coach**



|  |
| --- |
| Fig 2.2.4 |

**Examples of public gym routines available in the gym**

## Log

|  |  |  |
| --- | --- | --- |
| Date | What did you do | Notes |
| February to March 2017 | Begin the project!   1. Brain storm ideas for project 2. Choose project 3. Problem Identification 4. Current System 5. Interview Client 2 (Intermediate) 6. Get evidence for current systems 7. Objectives |  |
| 06/04/17 | Completed Identification of prospective users.  Acceptable Limitations  Outline of proposed solutions | Need to refine objectives so aligns with Smart Targets |
| 17/04/17 | Client 1 (Beginner) Interview  Client 3 (Advanced) Interview |  |
| 20/4/17 | Got more evidence: Coach’s plan + Public Gym routines |  |
| 10/5/17 | Tabulated results from google form questionnaire |  |
| 11/5/17 | Made improvements to  -Current System  -Problem Identification |  |
| 14/5/17 | -Investigation Techniques  -Proposed Solution |  |
| 25/5/17 | Entity Relationship Diagram |  |
| 30/07/17 | Checked Coursework |  |
| 25/8/17 | Remade ERD diagram |  |
| 1/9/17 – 1/11/17 | * Work on Design * Work on planning code * Implementing Code * Wiring Pages together |  |

## Coding

|  |
| --- |
| Class: DB |
| ***Code***  <?php  Class DB{  private $\_query, $\_results, $\_count, $\_pdo;  private static $\_instance;  function \_\_construct(){  $this->\_pdo = new PDO('mysql:host=127.0.0.1;dbname=gymplan', 'hogrider','PBF5szWTUcznwbXz');  //By providing a password and username this adds security to database,  }  public static function getInstance(){  if(!isset(self::$\_instance)){  self::$\_instance = new DB();  }  return self::$\_instance;  }    public function prep($query) {    return $this -> \_pdo -> prepare($query);  }    public function query($sql, $params = array()){  $this->\_query = $this->\_pdo->prepare($sql);  if(count($params)){  $i = 1;  foreach($params as $param){  $this->\_query->bindValue($i, $param);  $i++;  }  }  $this->\_query->execute();  $this->\_results = $this->\_query->fetchAll(PDO::FETCH\_OBJ);  $this->\_count = $this->\_query->rowCount();  return $this;  }  public function results(){  return $this->\_results;  }  public function count(){  return $this->\_count;  }  }  ?> |

|  |
| --- |
| Class: Exercise |
| ***Code***  <?php  Class Exercise{  private $eId, $name, $description, $image;  private $rId, $rest, $sets, $reps,$orderNum;  public function \_\_construct($eId,$rId){  $this->\_rId = $rId;  $this->\_eId = $eId;  $this->find($eId); // Populates general properties about an exercise  $this->exerciseDetails(); // Populates properties specific to exercise in a certain routine  }  private function find($eId){  $query = DB::getInstance()->prep("SELECT \* FROM exercise WHERE ExerciseId = ?");  $query->bindParam(1,$eId);  $query->execute();  $results=$query->fetchAll(PDO::FETCH\_OBJ);  $this->\_name = $results[0]->Name;  $this->\_description = $results[0]->Description;  $this->\_image = $results[0]->Image;  }  public function exerciseDetails(){  $query = DB::getInstance()->prep("SELECT \* FROM exercise2routine WHERE ExerciseId = ? AND RoutineId = ?");  $query->bindParam(1,$this->\_eId);  $query->bindParam(2,$this->\_rId);  $query->execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  $this->\_rest = $results[0]->RestBetween;  $this->\_sets = $results[0]->Sets;  $this->\_reps = $results[0]->Reps;  $this->orderNum = $results[0]->OrderInRoutine;  }  public function getName(){  return $this->\_Name;  }  public function getImage(){  $query = DB::getInstance()->prep("SELECT Image FROM exercise WHERE ExerciseId=?");  $query -> bindValue(1,$this->\_eId);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if ($results[0]->Image <> ""){  echo "<div id='img\_div'>";  echo "<img src='exerciseImages/".$results[0]->Image."' height='40%' width='32%'>"; //Uses path file and name of image from db to print image  echo "</div>";  }else{  echo "No Image Available";  }  }  //a set of functions that returns attributes of an object  public function getDescription(){  return $this->\_description;  }  public function getSets(){  return $this->\_sets;  }  public function getReps(){  return $this->\_reps;  }  public function getRest(){  return $this->\_rest;  }  public function getOrderInRoutine(){  return $this->\_orderNum;  }  public function getExerciseId(){  return $this->\_eId;  }  public function getMuscles($output){ // Returns a string of muscle id seperated by commas  $eId = $this->\_eId;    $query = DB::getInstance()->prep("SELECT \* FROM muscle  INNER JOIN muscle2exercise on muscle2exercise.MuscleId = muscle.MuscleId  WHERE muscle2exercise.exerciseId=?  ");  $query -> bindValue(1,$eId);  $query->execute();  $result\_array = $query->fetchall(PDO::FETCH\_ASSOC);  $stringMusclesId = "";  $stringMusclesName = "";  if ($output == 'id'){  foreach($result\_array as $result){  $stringMusclesId = $result['MuscleId'].','.$stringMusclesId;  }  return $stringMusclesId;  }else{ //$output ='name'  foreach($result\_array as $result){  $stringMusclesName = $result['Name']."<br>".$stringMusclesName;  }  return $stringMusclesName;  }    }  }  ?> |

|  |
| --- |
| Class: Routine |
| **Code**  <?php  Class Routine{  private $numOfExercise, $rId, $eId, $creator, $name;  private $exerciseArray = array();  private $weightButton,$submitButton, $feedbackButton;  private $recordId;  private $lastPerformed;  private $weight;  private $doneBefore;  private $musclesTrained = array();  public $e2rId;  private $timeTableId;  private $prevTimeTableId;  private $arrayMuscle = array();  private $arrayMuscleNames = array();  public function \_\_construct($rId,$timeTableId){  /\*  The Routine Id will be found before creating a new instance of routine  \*/  $this->findNumEx($rId);  $this->find($rId);  $this->\_rId = $rId;;  $this->\_timeTableId = $timeTableId;    for ($i=1; $i<=$this ->\_numOfExercise; $i++){  $this->findExerciseDetails($i,$rId); // Finds the exercise id given the order number and routine id  $this->\_exerciseArray[$i] = New Exercise($this->\_eId, $rId); // Stores each object of exercise in an array  // composition aggregation  }    }  public function printFinishRoutine(){  // Create a form, to input whether the routine was completed and the rating  echo "<form method='post'>";  echo "Completed";  echo "<input type='radio' value='Completed' name='Completed'>";  echo "<br>";  echo "Rating: ";  echo "<input type ='number' placeholder ='0-10' max='10' min='0' name='Rating' >";  echo "<br>";  echo "<input type ='submit' value='CompletedRoutine' name='CompleteCheck'>";  echo "</form>";  if (isset($\_POST['Completed']) && isset($\_POST['Rating']) && isset($\_POST['CompleteCheck'])){  $this->completeRoutine($\_POST['Rating']);  }  }  private function completeRoutine($rating){  $query = DB::getInstance()->prep("UPDATE timetable SET Completed = ?, Rating = ? WHERE UserId = ? AND RoutineId = ? AND SetDate = ?");  $query -> bindValue (1, 1);  $query -> bindValue(2, $rating);  $query -> bindValue(3, $\_SESSION['id']);  $query -> bindValue(4, $this->\_rId);  $query -> bindValue(5, $this->getCurrentDate());  $query -> execute();  //print\_r($query);  }  public function findAverageRating(){  $rId = $this->\_rId;  $query = DB::getInstance()->prep("SELECT AVG(Rating) as average FROM timetable WHERE RoutineId=? AND Completed='1' ");  $query -> bindValue(1,$rId);  $query -> execute();  $results=$query->fetchAll(PDO::FETCH\_OBJ);  return $results[0]->average;  // average rating is calculated on run time, rather than being stored in the database. This is necessary since the data is dynamic and it ensures no redundant data will be stored.  }  private function find($rId){  $query = DB::getInstance()->prep("SELECT \* FROM routine WHERE RoutineId = ?");  $query->bindParam(1,$rId);  $query->execute();  //print\_r($query);  $results = $query->fetchAll(PDO::FETCH\_OBJ);  $this->\_name = $results[0]->Name;  $this->\_creator = $results[0]->Creator;      }      private function findExerciseDetails($orderNum,$rId){  // Retrieves exerciseid  $query= DB::getInstance()->prep("SELECT \* FROM exercise2routine WHERE OrderInRoutine = '$orderNum' AND RoutineId='$rId' ");  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if (count($results)){  $this ->\_eId = $results[0]->ExerciseId;  }  }    private function findNumEx($rId){  // finds the number of exercises belonging to 1 routine  $query= DB::getInstance()->prep("SELECT COUNT(ExerciseId) AS num FROM exercise2routine WHERE RoutineId='$rId'"); //aggrgegate function  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  $this ->\_numOfExercise = $results[0]->num;  }  public function displayRoutine(){  //Function prints a routine  echo "<table border='1px'>";  echo "<tr bgcolor='#2ECC71'>";  echo "<td>";  echo "Name of Routine: ";  echo "</td>";  echo "<td>";  echo "<strong>".$this->\_name."</strong>";  echo "</td>";  echo "</tr>";    echo "<tr bgcolor='#AED6F1'>";  echo "<td>";  echo "Creator: ";  echo "</td>";  echo "<td>";  echo "<i>".$this->\_creator."</i>";  echo "</td>";  echo "</tr>";  echo "</table>";  // Prints a Routine, with no suggested weight/ prints suggested weight if routine is done before  // temporary test for loop  $count = 0;  foreach ($this->\_exerciseArray as $exercise){  $this->\_recordId = -1;  // Record Id is set to -1 at the beginning of every new exercise  $count = $count + 1;  if ($this->\_timeTableId<>-1){ // timetable id is -1, when the user is simply viewing the routine not in timetable  if ($this->\_doneBefore==true){  ?>  <html>  <table border='1px' id="routineTable">  <tr>  <th> Suggested Starting Weight </th>  </tr>  </html>  <?php  //SQL search for date, set, weight, uid    echo '<td>';  echo "Suggested Starting Weight for ".$exercise->getName()." is : ";  $this->generateSuggestedWeight($exercise->getExerciseId(),$exercise->getSets()); //Generates suggested weight based on previous feedback  echo '</td>';  ?>  <html>  </table>  </html>  <?php  }  }  ?>  <html>  <table border='1px' id="routineTable">  <tr bgcolor="#666699">  <th>Name</th>  <th>Description</th>  <th>Muscles</th>  <th>Image</th>  <th> Reps per Set</th>  <th> Rest Period </th>  </tr>  </html>  <?php  echo "<br>";  echo '<tr bgcolor ="#a3a3c2">';  echo '<td>'.$exercise->getName().'</td>';  echo '<td>'.$exercise->getDescription().'</td>';  echo '<td>'.$exercise->getMuscles('name').'</td>';  echo '<td>'.$exercise->getImage().'</td>';  echo '<td>'.$exercise->getReps().'</td>';  echo '<td>'.$exercise->getRest().'</td>';  echo '</tr>';  if($this->\_timeTableId<>-1){ // When timetableid = -1, it is when the routine is viewed from search so no feedback table  ?>  <html>  </table>  <table border='1px' id="feedbackTable">  </html>  <?php  echo '<br>';  echo '<tr>';      for ($i=1; $i<=$exercise->getSets(); $i++){  echo '<td>'.'Set No. '.$i;  $this->\_feedbackButton = $i."f".$exercise->getExerciseId();  $this->\_weightButton = $i."w".$exercise->getExerciseId();  $this->\_submitButton = $i."s".$exercise->getExerciseId();  //Feedback, Weight and Submit are given a unique name, combination of set number, f/w/s, exerciseId  ?>  <html>  <body>  <form method="post">  <select name="<?php echo htmlspecialchars($this->\_feedbackButton) ?>">  <option value=1> Easy </option>  <option value=2> Challenging </option>  <option value=3> Too Difficult </option>  </select>  <?php  // To check whether a user has performed a set, check the entry in the feedback table  // Infomation required is 1. RecordId and 2. Set Number  //checkInputForm checks if feedback has already been entered for a specific set  $this->findRecordId($exercise->getExerciseId()); //retrieves record id, so feedback can be inputted into feedbacktable  if($this->checkInputForm($i)){  //checkInputForm disables the form if the set has already been performed, prevent double entry.  //echo "Form Checked";  //if already inputted disable form  $placeText = "Record Entered";  $disabled = "disabled";  }else{  $placeText = "Enter Weight";  $disabled = "required";  }  ?>  Weight: <input type='text' name="<?php echo htmlspecialchars($this->\_weightButton) ?>" placeholder='<?php echo $placeText ?> ' <?php echo $disabled ?>>  <input type='submit' name="<?php echo htmlspecialchars($this->\_submitButton) ?>" value='submit' class='btn'>  <?php    // if boolean = true then echo "disabled" else echo "required"      if (isset($\_POST[$this->\_submitButton])){  if ($i==1){  // inputs data into record table once per exercise and finds recordId, needed for input feedback  $this->inputRecordTable($exercise->getExerciseId());  }  $this->findRecordId($exercise->getExerciseId());  $feedback = $\_POST[$this->\_feedbackButton];  $weight= $\_POST[$this->\_weightButton];  $this->inputFeedbackTable($feedback,$weight,$i);  echo "Your suggested weight is: ";  echo ($this->implementFeedback($feedback,$weight,$exercise->getReps()));  }  ?>  <html>  </form>  </body>  </html>  <?php echo '</td>';  }  echo '</tr>';  ?>  <html>  </table>  </html>  <?php  }  }  }    private function inputFeedbackTable($feedback,$weight,$currentSet){  //Input Set Number, Feedback, Weight, RecordId  $recordId = $this->\_recordId;  //echo "recordId is: ".$recordId;  $query= DB::getInstance()->prep("INSERT INTO feedback (SetNumber, Feedback, Weight, RecordId) VALUES (?,?,?,?)");  //print\_r($query);  $query -> bindValue(1,$currentSet);  $query -> bindValue(2,$feedback);  $query -> bindValue(3,$weight);  $query -> bindValue(4,$recordId);  $query -> execute();    }  private function inputRecordTable($exerciseId){  //Input RoutineId, UserId, ExerciseId, Date    $userId = $\_SESSION ['id'];  $this->getE2RId($exerciseId);  $query= DB::getInstance()->prep("INSERT INTO record (TimeTableId, E2RId) VALUES (?,?)");  $query -> bindValue(1, $this->\_timeTableId);  // timetableId is passed into new Routine from timetable class  // it can be stored as a property in routine class and passed into this function  $query -> bindValue(2, $this ->\_e2rId);  //e2rId can be retrieved within this class  $query -> execute();    }  private function findRecordId($exerciseId){  $this->getE2RId($exerciseId);  $timeTableId = $this->\_timeTableId;  $query = DB::getInstance()->prep("SELECT \* FROM record WHERE E2RId = '$this->\_e2rId' AND TimeTableId= '$this->\_timeTableId' ");  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);    if (count($results)){  $this->\_recordId = $results[0]->RecordId;  }else{  $this->\_recordId = -1;  }  }  private function findPreviousRecordId($exerciseId){  $this->getE2RId($exerciseId);  if ($this->\_doneBefore == true){  // If performed before, find previous timetable id to look for previous feedback  $this->\_prevTimeTableId = $this->getPreviousTimeTableId();    $query = DB::getInstance()->prep("SELECT \* FROM record WHERE E2RId = '$this->\_e2rId' AND TimeTableId= '$this->\_prevTimeTableId' ");  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if (count($results)){  $this->\_recordId = $results[0]->RecordId;  }  }else{  }  }  private function getPreviousTimeTableId(){  //Find previous timetable id, the timetable id that belongs to the user when they last performed the routine  $uId = $\_SESSION['id'];  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE SetDate='$this->\_lastPerformed' AND UserId ='$uId' AND RoutineId='$this->\_rId' ");  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  return $results[0]->TimeTableId;    }    public function checkRoutineDoneBefore(){    /\*  Search TimeTable class to see if there is a record of matching UserId and RoutineId  Take into account of date, for example if routine done too long ago, don't use data: user likely to undergo reversiblity hence cant maintain same intensity  \*/  $rId = $this->\_rId;  $uId = $\_SESSION['id'];  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE RoutineId=? AND UserId=? AND Completed='1' GROUP BY SetDate desc");  $query -> bindValue(1, $rId);  $query -> bindValue(2, $uId);  $query->execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if (count($results)){  //echo "<br>";  $this->\_lastPerformed = $results[0]->SetDate;  //echo " Last Performed: ".$this->\_lastPerformed;  //echo '<br>';  $now = $this->getCurrentDate();  $date1 = new DateTime($this->\_lastPerformed);  $date2 = new DateTime($now);  $diff = $date2->diff($date1)->format("%a");  if ($diff<=31){  $this->\_doneBefore = true;  return true;  }else{  $this->\_doneBefore = false;  // if the routine was done more than a month ago return false  return false;  }  }else{  $this->\_doneBefore = false;  return false;  }    }  private function getCurrentDate(){  date\_default\_timezone\_set('Singapore');  $date = date('Y-m-d ');  //$date = date('Y/m/d H:i:s'); Gets Hour, Minutes, Seconds  return $date;  }  private function generateSuggestedWeight($eId,$sets){  //Function generates a suggested weight or a range of weights based on the user’s feedback  //Feedback function retrieves the feedback given by the user, from the most recent time the routine was performed  //It takes challenging weight as the highest priority, hence takes highest challenging weight as suggested weight, if that is not possible it takes the lowest too difficult weight and/or the highest easy weight  $challengingWeight = 0;  $lowerLimit = 0;  $upperLimit = 1000000;  for ($i=1; $i<=$sets; $i++){  $feedback = $this->getFeedback($eId,$sets,$i);  if ($feedback==2){ // 2 = challenging  if ($challengingWeight<$this->\_weight){  $challengingWeight = $this->\_weight;  // Sets the highest possible challenging weight  }  }elseif($feedback == 1){ // 1 = too easy  if($lowerLimit < $this->\_weight){  $lowerLimit = $this->\_weight;  // Sets the highest possible easy weight  }  }else{ //feedback = too difficult  if($upperLimit > $this->\_weight){  $upperLimit = $this->\_weight;  // Sets the lowest possible too difficult weight  }  }  }  if ($challengingWeight > 0){  echo $challengingWeight."kg";  }elseif($lowerLimit<$upperLimit){  if($upperLimit<1000000 && $lowerLimit>0){  echo "Between ".$upperLimit."kg and".$lowerLimit."kg";  }elseif($lowerLimit==0){  echo "Below ".$upperLimit."kg";  }else{  echo "Above ".$lowerLimit."kg";  }  }else{  echo "No Weight can be suggested";  }    }  private function getFeedback($eId,$setNo,$currentSet){  // Use last performed date, eId, userId, routineId to find RecordId  // Use RecordId, SetNumber to find Weight AND Feedback  $this->findPreviousRecordId($eId); //Finds previous recordid to find previous feedback  $query = DB::getInstance()->prep("SELECT \* FROM feedback WHERE recordId='$this->\_recordId' AND SetNumber='$currentSet' ");  $query->execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if (count($results)){  $feedback = $results[0]->Feedback;  $this->\_weight = $results[0]->Weight;  /\*  echo "Your feedback was ".$feedback;  echo "<br>";  echo "Weight was".$this->\_weight;  \*/  return $feedback;  }    }  private function implementFeedback($feedback,$weight,$reps){  //generates a suggested weight based on the feedback  // For exercises with a rep range of 6 or below, add increments of 5kg and decrease by 2.5kg  // For exercises with a rep range greater than 6, change by increments of 2.5kg  if($feedback == 2){  return $weight;  }elseif($feedback == 1){  if ($reps <=6){  $SuggestedWeight = $weight + 5;  } elseif ($reps > 6){  $SuggestedWeight = $weight + 2.5;  }    }elseif($feedback == 3){  if ($reps <=6){  $SuggestedWeight = $weight - 2.5;  } elseif ($reps > 6){  $SuggestedWeight = $weight - 2.5;  }  }  return $SuggestedWeight;    }  public function setDetails(){  // Gets Muscles Trained in a list seperated by commas  $muscleString = $this->getMusclesTrained();  // Turn string seperated by commas into array  $this->\_arrayMuscle = explode(',', $muscleString);  // Make sure all values in array are unique, so only 1 muscle of each type can be outputted  $this->\_arrayMuscle = array\_unique($this->\_arrayMuscle);  $count = -1;  foreach($this->\_arrayMuscle as $muscle){  $count = $count +1;  $query = DB::getInstance()->prep("SELECT Name FROM muscle WHERE MuscleId=?");  $query -> bindValue(1, $muscle);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  $this->\_arrayMuscleNames[$count] = $results[0]->Name;    }  }  public function getRoutineName(){  return $this->\_name;  }  public function getMusclesTrained(){  // using muscle 2 exercise, input the muscles trained into the array , ensuring no repeats  // each exercise class holds the property of muscles trained  $count = 0;  foreach($this->\_exerciseArray as $exercise){  $this->musclesTrained[$count] = $exercise->getMuscles('id');  // Array holding a string of muscles id seperated by commas, with each position in the array holding infomation about the muscles trained for that corresponding exercise  $count = $count + 1;  }  $muscleString="";  for($i=0; $i<$this->\_numOfExercise;$i++){  $muscleString = $this->musclesTrained[$i].$muscleString;  // Turns array into a long string of muscle ids, which end in a comma  }  // Remove last comma  $length = strlen($muscleString);  $muscleString = substr\_replace($muscleString,"",$length-1,$length);    return $muscleString;    }  private function getE2RId($exerciseId){  $query = DB::getInstance()->prep("SELECT \* FROM exercise2routine WHERE RoutineId = ? AND ExerciseId = ?");  $query -> bindValue(1, $this->\_rId);  $query -> bindValue(2, $exerciseId);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  //print\_r($results);  $this->\_e2rId = $results[0]->E2RId;  }  public function getLastPerformedDate(){  return $this->\_lastPerformed;  }  public function getRoutineId(){  return $this->\_rId;  }  public function routinePerformedBefore(){  return $this->\_doneBefore;  }  public function checkMuscle($muscleSearch){  //Checks if muscle is in routine  foreach($this->\_arrayMuscleNames as $muscle){  if($muscle == $muscleSearch){  return true;  }  }  return false;  }  public function getMusclesRoutine(){  foreach($this->\_arrayMuscleNames as $muscle){  echo "<br>";  echo $muscle;  }  }  private function checkInputForm($setNumber){  if ($this->\_recordId<>-1){  //if recordId has been set, set 1 has been performed  //recordid must be used to deterine if other sets have been performed  $query = DB::getInstance()->prep("SELECT \* FROM feedback WHERE RecordId = ? AND SetNumber = ?");  // NEED to use RECORDID  // ERROR Disables future forms  $query -> bindValue(1, $this->\_recordId);    $query -> bindValue(2, $setNumber);  $query -> execute();  $results = $query ->fetchAll(PDO::FETCH\_OBJ);    if (count($results)){  //echo "Set ".$setNumber. "has been performed";  return true;  }else{  //echo "Set ".$setNumber." has not been performed";  return false;  }  }else{  //echo "Record Id not set";  return false;  }  }  }  ?> |

|  |
| --- |
| Class: Search |
| ***Code***  <?php  Class Search{  private $routineTemp = array();  private $sortVar;  private function sortRating(){  // Sorts routineTemp() based on sortVar in descending order  // Highest Rated First / Most Recently Performed First  for ($i=0;$i<count($this->\_routineTemp);$i++){  for ($j=0;$j<count($this->\_routineTemp)-1;$j++){  if($this->\_routineTemp[$j]->findAverageRating()<$this->\_routineTemp[$j+1]->findAverageRating()){  $temp = $this->\_routineTemp[$j];  $this->\_routineTemp[$j]=$this->\_routineTemp[$j+1];  $this->\_routineTemp[$j+1]=$temp;  }  }  }  }  private function sortDate(){  for ($i=0;$i<count($this->\_routineTemp);$i++){  for ($j=0;$j<count($this->\_routineTemp)-1;$j++){  if ($this->\_routineTemp[$j]->checkRoutineDoneBefore() == true && $this->\_routineTemp[$j+1]->checkRoutineDoneBefore() == true){  //If both routines have been performed before, compare the dates to see which is more recent  if(strtotime($this->\_routineTemp[$j]->getLastPerformedDate()) < strtotime($this->\_routineTemp[$j+1]->getLastPerformedDate())){  //Conversion of data type  $temp = $this->\_routineTemp[$j];  $this->\_routineTemp[$j]=$this->\_routineTemp[$j+1];  $this->\_routineTemp[$j+1]=$temp;  }  }elseif($this->\_routineTemp[$j]->checkRoutineDoneBefore() == true && $this->\_routineTemp[$j+1]->checkRoutineDoneBefore() ==false){  // do nothing, allow a routine that has been done before to be placed ahead of one which is not done  }elseif($this->\_routineTemp[$j]->checkRoutineDoneBefore() == false && $this->\_routineTemp[$j+1]->checkRoutineDoneBefore() ==true){  // switch it around  $temp = $this->\_routineTemp[$j];  $this->\_routineTemp[$j]=$this->\_routineTemp[$j+1];  $this->\_routineTemp[$j+1]=$temp;  }    }  }  }  public function searchRoutine($name, $sortVar, $searchParam){  //Searches for routine based on the search parameter (name, creator or muscle group), which takes a string as input, and sorts the routines based on sort variable (date performed or rating)  $query = DB::getInstance()->prep("SELECT \* FROM routine WHERE $searchParam LIKE '%$name%'");  $query -> execute();  print\_r($query);  $results= $query->fetchALL(PDO::FETCH\_ASSOC);  if(count($results)){  $count = -1;  $this->\_sortVar = $sortVar;  foreach($results as $result){  $count = $count + 1;  $routineId = $result['RoutineId'];  $this->\_routineTemp[$count] = new Routine($routineId,-1); // Association Aggregation: Each routine found is instantiated and stored into an array    }  if ($sortVar == 'Rating'){  //Both sortRating and sortDate sort the routine using a bubble sort  $this->sortRating();  }else{  $this->sortDate();  }    for($i=0; $i<=$count; $i++){  echo "<br>";  $this->\_routineTemp[$i] ->displayRoutine();  echo "<br>";  echo "<table border='1px'>";  echo "<tr>";  echo "<td>";  echo "Average rating is: ";  echo "</td>";  echo "<td>";  echo $this->\_routineTemp[$i]->findAverageRating();  echo "<br>";  echo "</td>";  echo "</tr>";  if ($this->\_routineTemp[$i]->checkRoutineDoneBefore()==true){  echo "<tr>";  echo "<td>";  echo "Last performed date ";  echo "</td>";  echo "<td>";  echo $this->\_routineTemp[$i]->getLastPerformedDate();  echo "</td>";  echo "<br>";  echo "</tr>";  }else{  echo "<tr>";  echo "<td>";  echo "Never performed before ";  echo "</td>";  echo "</tr>";  }  echo "</table>";  echo "<br>";  }  }else{  return false;  }      }  public function searchRoutineMuscle($muscleSearch, $sortVar){  $this->\_sortVar = $sortVar;  $query = DB::getInstance()->prep("SELECT \* FROM routine");  $query -> execute();  $results = $query -> fetchALL(PDO::FETCH\_ASSOC);  $muscleInRoutine = false;  if(count($results)){  $count = -1;  foreach($results as $result){    $routineId = $result['RoutineId'];  $tempRoutine = new Routine($routineId, -1);  $tempRoutine -> setDetails();    if($tempRoutine->checkMuscle($muscleSearch)){ // See if muscle is in routine  $muscleInRoutine = true;  $count = $count + 1;  $this->\_routineTemp[$count] = new Routine($routineId, -1);  }  }  if ($sortVar == 'Rating' && $muscleInRoutine == true){  $this->sortRating();  }elseif ($sortVar == 'DatePerformed' && $muscleInRoutine == true){  $this->sortDate();  }else{  echo "<br>";  echo "No results found ";  echo "<br>";  }    for($i=0; $i<=$count; $i++){  echo $result['Name'];  $this->\_routineTemp[$i] ->displayRoutine();  }  }  }  ?> |

|  |
| --- |
| Class: Timetable |
| ***Code***  <?php  class TimeTable{  private $daysArray = array();  private $dateArray = array();  public function \_\_construct(){  $this->generateTimeTable(); //prints a timetable with routines in each day  $this->displayAddRoutine(); //adds a button for user to add routine  $this->displayRemoveRoutine(); // adds a button for user to remove routine    if(isset($\_POST['AddRoutine'])){  $date = $\_POST['date'];  $routineId = $\_POST['routine'];  $time = $\_POST['time'];  echo $date;  echo "<br>";  echo $routineId;  echo "<br>";  echo $time;  echo "<br>";  // Checks if rthere is a routine on that day  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE SetDate=? AND UserId=?");  $query -> bindValue(1,$date);  $query -> bindValue (2, $\_SESSION['id']);  $query -> execute();  print\_r($query);  echo "<br>";  $results = $query->fetchAll(PDO::FETCH\_OBJ);  print\_r($results);    if(count($results)){    echo "There is already a routine, please remove it before adding a new one. ";  }else{  echo "You have added a routine on".$date;  $this->insertTimetableDay($date, $routineId, $time);  }  }  if(isset($\_POST['RemoveRoutine'])){  $date = $\_POST['date'];  echo $date;  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE SetDate=? AND UserId =?");  $query -> bindValue(1,$date);  $query -> bindValue (2, $\_SESSION['id']);  $query -> execute();  echo "<br>";  print\_r($query);  echo "<br>";  $results = $query->fetchAll(PDO::FETCH\_OBJ);  print\_r($results);    if(count($results)){  echo "Removed routine on ".$date;  $this->removeTimetableDay($date);  }else{  echo "There is no routine on this day!";  }    }    }  private function removeTimetableDay($date){  $query = DB::getInstance()->prep("DELETE FROM timetable WHERE SetDate=? AND UserId=?");  $query -> bindValue(1,$date);  $query -> bindValue(2,$\_SESSION['id']);  $query -> execute();  }  private function displayRemoveRoutine(){    ?>  <form method="post">  <select name= "date">  <?php  for ($i=0 ; $i<=6; $i++){  echo "<option value= ";  echo $this->dateArray[$i];  echo ">";  echo $this->dateArray[$i];  echo "</option>";  }  ?>  <input type='submit' name='RemoveRoutine' class='btn' value="Remove Routine">  </select>  </form>  <?php  }  private function insertTimetableDay($date, $routineId, $time){  $query = DB::getInstance()->prep("INSERT into timetable (UserId, RoutineId, SetTime, SetDate) VALUES (?,?,?,?)");  $query -> bindValue(1, $\_SESSION['id']);  $query -> bindValue(2, $routineId);  $query -> bindValue(3, $time);  $query -> bindValue(4, $date);  $query -> execute();  }  public function displayAddRoutine(){  ?>  <form method="post">  <?php  $query = DB::getInstance() -> prep("SELECT \* FROM routine ");  $query -> execute();  $results = $query->fetchALL(PDO::FETCH\_ASSOC);  echo "<select name='routine'>";  if(count($results)){  foreach ($results as $result){    echo "<option value=" ;  echo $result['RoutineId'];  echo ">";  echo "Name: ".$result['Name'];  echo "</option>";  }  echo "</select>";  }  ?>  //Prints a drop down list of routine names  <select name= "date">  <?php  for ($i=0 ; $i<=6; $i++){  echo "<option value= ";  echo $this->dateArray[$i];  echo ">";  echo $this->dateArray[$i];  echo "</option>";  }  ?>  </select>  //Prints a drop down list of dates of the week  Time: <input type='time' name='time' required>  //Time Data type required, prevent wrong format input of data  <input type='submit' name='AddRoutine' class='btn' value='Add Routine'>  </form>  <?php  }  private function generateTimeTable(){  $this->\_daysArray= array('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday');  $this->populateDates();  ?>  <table border = '1px'>  <tr>  <?php  for($i='0'; $i<='6'; $i++){    echo "<br>";  if ($this->dateArray[$i] == $this->getDate()){  echo "<th bgcolor = '#66ffff'>"; //Prints today's date in blue  }else{  echo "<th>";  }  echo $this->dateArray[$i];  echo "<br>";  echo $this->\_daysArray[$i];  echo "</th>";  }  echo "</tr>";  echo "<tr>";  for($i='0'; $i<='6'; $i++){  if ($this->checkRoutinePresent($this->dateArray[$i])){  // print Routine Name, Time Started, Muscle Groups  }else{  // prints empty cell  echo "<td width =250px>";  echo "<br>";  echo " No Routine ";  echo "</td>";  }  }  echo "</tr>";  echo "</table>";  }  private function checkRoutinePresent($date){  $uId = $\_SESSION['id'];  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE UserId= '$uId' AND SetDate= '$date'");  //print\_r($query);  $query -> execute();  echo "<br>";  //print\_r($query);  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if (count($results)){  // Print Routine Details For That Day  echo "<td width =250px>";  echo "<br>";  echo "Set Time: ".$results[0]->SetTime;  echo "<br>";  echo "Set Date: ".$results[0]->SetDate;  echo "<br>";  $routineId = $results[0]->RoutineId;  $timeTableId = $this->findTimeTableId($date,$uId,$routineId);  $routine = New Routine($routineId,$timeTableId); // Association Aggregation    $routine->setDetails();  echo "<strong>".$routine->getRoutineName()."</strong>";  echo $routine->getMusclesRoutine();    echo "<br>";  echo "Average Rating: ".$routine->findAverageRating($routineId);  if ($results[0]->Completed == 1){  echo "<br>";  echo "<strong>";  echo "Routine Completed! ";  echo "</strong>";  echo "<br>";  }else{  echo "<br>";  echo "<strong>";  echo "Yet to do ";  echo "</strong>";  echo "<br>";  }  echo "</td>";  return true;  }  }  private function findTimeTableId($date,$uId,$routineId){  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE UserId=? AND SetDate=? AND RoutineId=? ");  $query -> bindValue(1,$uId);  $query -> bindValue(2,$date);  $query -> bindValue(3,$routineId);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  return $results[0]->TimeTableId;  }  private function populateDates(){  // Gets the date of the specific week day e.g Wednesday = 4th November  $currentDate = $this->getDate();  $day = date('N', strtotime($currentDate)); //Mon = 1 Sun = 7  $this->dateArray[$day-1]=$currentDate;  for($i=0;$i<$day-1;$i++){  $daysDiff = $day - $i-1;  // Subtract Current Date and days Diff  $newDate = date('Y-m-d',strtotime('-'.$daysDiff.'day', strtotime($currentDate)));  $this->dateArray[$i] = $newDate;  }  for($i = $day; $i<=6; $i++){  $daysDiff = $i - $day + 1;  // Add Current Date and days Diff  $newDate = date('Y-m-d',strtotime('+'.$daysDiff.'day', strtotime($currentDate)));  $this->dateArray[$i] = $newDate;  }  }  private function getDate(){  //date\_default\_timezone\_get();  date\_default\_timezone\_set('Singapore');  $date = date('Y-m-d ');    return $date;    }  }  ?> |

s

|  |
| --- |
| Class: User |
| ***Code***  <?php  Class User{  protected $firstName, $lastName, $dateOfBirth, $email, $uID;  public function \_\_construct($id = null){  // if no arguement is given to construct, $id is given the default value of null  //if(!isset($id)){  //check if logged in, if they are give $id their id as a value.    if($this->isLoggedIn()){  $id = $\_SESSION['id'];  if (isset($id)){  $this->find($id);  }  }      //}  //use find to retrieve the user's real data.  }  public function logOut(){  session\_start();  session\_destroy();  session\_unset();  header('Location: index.php');  }  public function isLoggedIn(){  return isset($\_SESSION['id'])? true: false;  // Checks if user id is stored in session  }  protected function find($id){  $query = DB::getInstance()->prep("SELECT \* FROM users WHERE UserId = ?");  $query->bindParam(1,$id);  $query->execute();  $results=$query->fetchAll(PDO::FETCH\_OBJ);    $this->firstName = $results[0]->FirstName;  $this->email = $results[0]->Email;  $this->lastName = $results[0]->LastName;  $this->dateOfBirth = $results[0]->DateOfBirth;  $this->email = $results[0] ->Email;  $this->salt =$results[0]->Salt;  $this->uID = $id;  //code to populate properties.  }  public function login($email, $password){  // Find user in database using their email, retrieve their corresponding salt  // Add salt to password, and hash iterator apply  // Compare hashed password with database  $query = DB::getInstance()->prep("SELECT \* FROM users WHERE Email = ?");  $query->bindValue(1,$email);  $query->execute();  $results=$query->fetchAll(PDO::FETCH\_OBJ);  if(count($results)){  // Hashes the password with the salt in the database  // Compares if hashed password is equal to that stored in db  // If equal user is logged in, user has its properties populated  // If not, log in is unsucessfull  if((hash('sha256',$password, $results[0]->Salt)) == ($results[0]->Password)){  $\_SESSION['id'] = $results[0]->UserId; //User Id is generated in database, and stored in session  $\_SESSION['name'] = $results[0]->FirstName;  return true;  }else{  echo "Wrong Password!";  }  }else{  echo "No Email Exists Please Register. ";  }  }  public function getEmail(){  return $this->email;  }  public function getFirstName(){  return $this->firstName;  }  public function getLastName(){  return $this->lastName;  }  public function getDateOfBirth(){  return $this->dateOfBirth;  }  public function getID(){  return $this->uID;  }  public function getSalt(){  return $this->salt;  }  public function changePassword($oldPass, $newPass, $email){  //This function is called when $oldPass = $newPass    if ($this->login($email,$oldPass)){  echo "correct password";  echo '<br>';  $salt = $this->getSalt();  echo $salt;  echo '<br>';  echo hash("sha256",$newPass,$salt);  echo '<br>';  echo '<br>';  $newPass = hash("sha256",$newPass,$salt);    $query = DB::getInstance() -> prep("UPDATE users SET Password = ? WHERE Email= ?");  //SQL Updates the password, at the corresponding record containing the email address  $query -> bindValue(1, $newPass);  $query -> bindValue(2, $email);  $query -> execute();  //var\_dump($query);      } else {  echo "Incorrect password";  }  }  public function resetPassword($newPassword,$email){  echo "Email is".$email;  echo "<br>";  echo "New Password is ".$newPassword;  $salt = $this->getSalt();  $saltPassword = hash("sha256",$newPassword,$salt);  echo "Salted Password :".$saltPassword;  echo "<br>";  echo "salt is ".$salt;    $query = DB::getInstance() -> prep("UPDATE users SET Password = ? WHERE Email= ?");  //"UPDATE users SET Password='$newPass' WHERE Username = '$username' AND Password = '$oldPass'"  $query -> bindValue(1, $saltPassword);  //$query -> bindValue(2, $salt);  $query -> bindValue(2, $email);  $query -> execute();  }  }  ?> |

|  |
| --- |
| Autoloader |
| <?php  session\_start();  spl\_autoload\_register(function($class){  require\_once('classes/'.$class.'.php');  });  ?> |

|  |
| --- |
| Page: addowner.php |
| ***Code***  <?php  require\_once('core/init.php');  $user = new User();  if(isset($\_POST['createOwner'])){    $dbConn = DB::getInstance();    $name = $\_POST["name"];  $email = $\_POST["email"];  $salt = mcrypt\_create\_iv(10);  $password = $\_POST["password"];  $confirmPassword = $\_POST["confirmPassword"];    if($password == $confirmPassword) {    $password = hash("sha256", $password, $salt);    if(!empty($name) && !empty($email) && !empty($password)) {    $query = $dbConn -> prep("INSERT INTO users (first\_name, email, password, salt, type) VALUES(?, ?, ?, ?, ?)");  // takes the input and changes the records in the data base for that specific user  $query -> bindValue(1, $name);  $query -> bindValue(2, $email);  $query -> bindValue(3, $password);  $query -> bindValue(4, $salt);  $query -> bindValue(5, 2);  $query -> execute();    header("Refresh: 1; url = home.php?");    } else {    echo "<p align=center>You have not properly filled the form</p>";  }    } else {    echo "Password aren't the same!";  }  //create worker account then log worker in.  }  ?>  <!DOCTYPE html>  <html>  <body>  <form method='post'>  <input type='text' name='name' placeholder='Name' required>  <input type='email' name='email' placeholder='Email' required>  <input type='password' name='password' placeholder='Password' required>  <input type='password' name='confirmPassword' placeholder='Confirm Password' required>  <input type='submit' name='createOwner' value='Create Account'>  </form>  </body>  </html> |
| ***Image*** |

|  |
| --- |
| Page: create.php |
| ***Code***  <html>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>Create</h1>  </div>  </html>  <?php  /\*  Create should allow Users to create a routine, by providing name, exercises, sets, reps drop down list  Two Forms will be used: 1. Enter Name and Number of Routines 2. Enter each exercise detail e.g reps,sets  \*/  require\_once "core/init.php";  include('ColumnButtons.html');  //Print First Form  ?>  <html>  <form method="get" action='create.php'>  Name of Routine: <input type='text' name='RoutineName' required> <br>  Number of Exercises: <input type='text' name='NumExercise' placeholder='0' required> <br>  <input type = "submit" value="Create a Routine!" class='btn' name='create'>  </form>  </html>  <?php  if(isset($\_GET['create'])){  $num = $\_GET['NumExercise'];  $nameRoutine = $\_GET['RoutineName'];  echo "<table border='1px' >";  echo "<tr>";  echo "<th>";  echo "Exercise";  echo "</th>";  echo "<th>";  echo "Sets";  echo "</th>";  echo "<th>";  echo "Reps";  echo "</th>";  echo "<th>";  echo "Rest Period (seconds)";  echo "</th>";  echo "</tr>";  echo "<form>";  echo "<input type='hidden' name='RoutineName' value='$nameRoutine'>";  echo "<input type='hidden' name='NumExercise' value='$num'>";  //Hidden type required to pass data to next form  for ($i=1; $i<=$num; $i++){  echo "<tr>";  //Prints a table with number of rows equal to number of exercises specified  echo "<td>";  printExercises($i);  echo "</td>";  echo "<td>";  echo "<input type='text' name='Sets".$i."' placeholder='0 Sets' required>";  echo "</td>";  echo "<td>";  echo "<input type='text' name='Reps".$i."' placeholder='0 Reps' required>";  echo "</td>";  echo "<td>";  echo "<input type='number' min='0' max='3600' name='Rest".$i."' required>";  echo "</td>";  //Ensures correct format when inputting data, reduce likelihood of extreme or erroneous inputs  echo "</tr>";  }  echo "</table>";  //echo "<form method='get' action='create.php'>";  echo "<input type='submit' name='Create' class='btn' value='Test'>";  echo "</form>";  }  if (isset($\_GET['Create'])){  echo "Created Routine!";  echo "<br>";  echo $\_GET['RoutineName'];  echo "<br>";  echo $\_SESSION['name'];  echo "<br>";  //Input Name and creator into Routine Table after first form has been successfully completed  $query = DB::getInstance()->prep("INSERT into routine (Name, Creator) VALUES (?,?) ");  $query -> bindValue(1, $\_GET['RoutineName']);  $query -> bindValue(2, $\_SESSION['name']);  $query -> execute();  print\_r($query);  echo "<br>";  $query = DB::getInstance()->prep("SELECT \* from routine WHERE Name = ? AND Creator= ? ");  $query -> bindValue(1, $\_GET['RoutineName']);  $query -> bindValue(2, $\_SESSION['name']);  $query -> execute();  $results = $query->fetchALL(PDO::FETCH\_ASSOC);  foreach($results as $result){  $routineId = $result['RoutineId'];  echo "Routine Id is ".$routineId;  for($i=1; $i<=$\_GET['NumExercise']; $i++){  if( isset($\_GET["ex".$i]) && isset($\_GET["Sets".$i]) && isset($\_GET["Reps".$i]) && isset($\_GET["Rest".$i]) ){  $exId = $\_GET["ex".$i];  $order = $i;  $sets = $\_GET["Sets".$i];  $reps = $\_GET["Reps".$i];  $rest = $\_GET["Rest".$i];  echo "ExerciseId is ".$exId;  echo "<br>";  echo "order is ".$order;  echo "<br>";  echo "ExerciseId is ".$exId;  echo "<br>";  echo "Order ".$order;  echo "<br>";  echo "Sets is ".$sets;  echo "<br>";  echo "Reps is ".$reps;  echo "<br>";  echo "Rest is ".$rest;  echo "<br>";  $query = DB::getInstance()->prep("INSERT into exercise2routine (RoutineId, ExerciseId, OrderInRoutine, Reps, Sets, RestBetween) VALUES (?,?,?,?,?,?) ");  $query -> bindValue(1, $routineId);  $query -> bindValue(2, $exId);  $query -> bindValue(3, $i);  $query -> bindValue(4, $reps);  $query -> bindValue(5, $sets);  $query -> bindValue(6, $rest);  $query->execute();  print\_r($query);    }  }  }  }  function printExercises($orderInRoutine){  $query = DB::getInstance()->prep('SELECT \* FROM exercise');  $query -> execute();  $results = $query->fetchALL(PDO::FETCH\_ASSOC);  if(count($results)){  //echo "<form method='post'>";  echo "<select name= 'ex".$orderInRoutine."' >";  foreach ($results as $result){  ?>  <option value="<?php echo $result['ExerciseId']; ?>"><?php echo $result['Name']; ?></option>  <?php  }  }  }  ?> |
| ***Image*** |

|  |
| --- |
| Page: home.php |
| ***Code***  <html></head>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>GYM PLANNER </h1>  <form method = "post" >  <input type ="submit" name = "logOut" value = "Log Out" class='btn' />  </div>  </form>  </html>  <?php  include('ColumnButtons.html');  require\_once('core/init.php');  $user = new User();  if(!$user->isLoggedIn()){  $user->logOut();  }    if(isset($\_POST['logOut'])){  $user->logOut();  }  echo "Hello ";  echo $user->getFirstName();  //echo "<br>";  //echo "User ID is: ".$\_SESSION ['id'];  //echo "<br>";  $date = todayDate();  checkRoutinePresent($date);  function todayDate(){  date\_default\_timezone\_set('Singapore');  $date = date('Y-m-d ');  return $date;  }  function checkRoutinePresent($date){  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE SetDate=? AND UserId=? AND Completed = '0'");  $query -> bindValue(1,$date);  $query -> bindValue(2, $\_SESSION['id']);  $query -> execute();  $results=$query->fetchAll(PDO::FETCH\_OBJ);    if(count($results)){  echo "<br>";  echo "Today's Routine";  echo "<br>";  $routine = new Routine($results[0]->RoutineId,$results[0]->TimeTableId);  $routine->checkRoutineDoneBefore();  $routine->displayRoutine();  $routine->printFinishRoutine();      }else{  echo "No Routine Today";  }    }  ?> |
| ***Image*** |

|  |
| --- |
| Page: index.php |
| ***Code***  <?php  require\_once('core/init.php');  $user = new User();  if($user->isLoggedIn()){  header('Location: home.php');  }  if(isset($\_POST['login'])){    if($user->login($\_POST['email'], $\_POST['password'])){    header('Location: home.php');  }  }  ?>  <html>  <head> <link rel='stylesheet' type='text/css' href='Style.css'> </head>  <body align = "center">  <title>  Gym Planner  </title>  <div id='topBanner'>  <h1> GYM PLANNER </h1>  </div>    <div id ='backgroundImage'>  <div id ='login'>  Please log in with your Gym Planner account  </div>  <form method='post'>  Username: <input type='text' name='email' placeholder='Email' required>  Password: <input type='password' name='password' placeholder='Password' required>  <input type='submit' name='login' value='Login' class='btn'>    </form>  <form method = "post" action = "register.php" >  Click here to register: <input type = "submit" value="register" class='btn'></p>  </form>    <form method = 'post' action='forgotpassword.php'>  <input type = 'submit' value = 'Forgot Password' class = 'btn'>  </div>  </body>  </html> |
| ***Image*** |

|  |
| --- |
| Page: log.php |
| <html>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>Log</h1>  </div>  </html>  <?php  require\_once "core/init.php";  include('ColumnButtons.html');  /\*  Log prints historcal data  - performed routines e.g Leg Routine 12/10/2017 and rating  \*/  printPastPerformedRoutines();  function printPastPerformedRoutines(){  $query = DB::getInstance()->prep("SELECT \* FROM timetable WHERE UserId=? AND Completed='1' GROUP BY SetDate desc");  $query -> bindValue(1, $\_SESSION['id']);  $query->execute();  $results = $query->fetchAll(PDO::FETCH\_ASSOC);    echo "<table border='1px'>";  echo "<tr>";  echo "<th>";  echo "Date performed";  echo "</th>";  echo "<th>";  echo "Your Rating";  echo "</th>";  echo "<th>";  echo "Routine Name";  echo "</th>";  echo "</tr>";  foreach($results as $result){  echo "<tr>";  echo "<th>";  echo $result['SetDate'];  echo "</th>";  echo "<th>";  echo $result['Rating'];  echo "</th>";  echo "<th>";  $query = DB::getInstance()->prep("SELECT \* FROM routine WHERE RoutineId = ? ");  $query -> bindValue(1, $result['RoutineId']);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_ASSOC);  foreach($results as $result){  echo $result['Name'];  }    echo "</th>";  echo "</tr>";  }  echo "</table>";  }  ?> |
| **Image** |

|  |
| --- |
| Page: logout.php |
| <?php  require\_once('core/init.php');  $\_SESSION['id'] = null;  header('Location: index.php');  ?> |

|  |
| --- |
| Page: profile.php |
| ***Code***  <!DOCTYPE>  <html>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>Profile</h1>  </div>  </html>  <?php  /\*  Create should allow Users to create a routine, by providing name, exercises, sets, reps drop down list  \*/  ini\_set('mysql.connect\_timeout',300);  ini\_set('default\_socket\_timeout',300);  require\_once"core/init.php";  include('ColumnButtons.html');  $user = new User();  if(!$user->isLoggedIn()){  header('Location: index.php');  }    $firstName = $user->getFirstName();  $lastName = $user->getLastName();  $dOB = $user->getDateOfBirth();  $email = $user->getEmail();  if(isset($\_POST['change'])){    $oldPass = $\_POST['OldPass'];  $newPass = $\_POST['NewPass'];  $confirmNewPass=$\_POST['ConfirmNewPass'];  echo $oldPass;  echo "<br>";  echo $newPass;  echo "<br>";  echo $confirmNewPass;  echo "<br>";  if($newPass == $confirmNewPass){ //double entry validation  if (!empty($oldPass) && !empty($newPass) && !empty($confirmNewPass)){ //entry validation  $user->changePassword($oldPass, $newPass, $email);  }    }  }  // PROFILE PICTURE  $uId = $\_SESSION['id'];  if(isset($\_POST['upload'])){  $msg="";    // the path to store the upload image  $target = "profileImages/".basename($\_FILES['image']['name']);  $image = $\_FILES['image']['name'];  $query = DB::getInstance()->prep("UPDATE users SET Image = ? WHERE UserId= ?");  $query -> bindValue(1, $image);  $query -> bindValue(2, $uId);  $query -> execute();  if (move\_uploaded\_file($\_FILES['image']['tmp\_name'], $target)){  $msg = "Upload Succesful";  echo $msg;  }else{  $msg = "There was a problem";  echo $msg;  }  }  $query = DB::getInstance()->prep("SELECT Image FROM users WHERE UserId=?");  $query -> bindValue(1,$uId);  $query -> execute();  $results = $query->fetchAll(PDO::FETCH\_OBJ);  if ($results[0]->Image <> ""){  echo "<div id='img\_div'>";  echo "<img src='profileImages/".$results[0]->Image."' height='40%' width='32%'>";  echo "</div>";  }  ?>  <html>  <div id = 'mainContent'>  <div id = 'subtitle'>  Personal Details  </div>  First Name: <?php echo $firstName; ?> </br>  Last Name: <?php echo $lastName; ?></br>  Date of Birth: <?php echo $dOB; ?> </br>  Email Address: <?php echo $email; ?></br>  </br>  To change password please enter original password</br>  <form method='post' >  Old Password: <input type='password' name='OldPass' placeholder='Old Password'/></br>  New Password <input type='password' name='NewPass' placeholder='New Password'/> </br>  Confirm New Password <input type='password' name='ConfirmNewPass' placeholder='New Password'/> </br>  <input type = 'submit' name = 'change' value='change'/>  </form>  <form method="post" action = "Profile.php" enctype="multipart/form-data">  <input type="hidden" name="size" value=1000000>  <br/>  <input type="file" name="image">  <br/><br/>  <input type="submit" name="upload" value="Upload Image"/>  </form>  </div>  </html> |
| ***Image*** |

|  |
| --- |
| Page: register.php |
| ***Code***  <?php  require\_once('core/init.php');  $user = new User();  if(isset($\_POST['Register'])){  // Form takes in the necessary inputs: Name, LastName, DoB, Password and it generates a salt  // If all fields have been filled out, these data inputs are then inserted into the database  // If there have been any errors due to presence or formatting, the form will reject the input of data and output a corresponding error message  $dbConn = DB::getInstance();  $firstName = $\_POST["FirstName"];  $lastName = $\_POST["LastName"];  $email = $\_POST["Email"];  $dob = $\_POST["DateOfBirth"];  $salt = mcrypt\_create\_iv(32);  $password = $\_POST["Password"];  $confirmPassword = $\_POST["confirmPassword"];  if($password == $confirmPassword) {    $password = hash("sha256", $password, $salt);    if(!empty($firstName) && !empty($lastName) && !empty($email) && !empty($password)) {    $query = $dbConn -> prep("INSERT INTO users (FirstName, LastName, Email, DateOfBirth, Salt, Password ) VALUES(?,?,?,?,?,?)");  // takes the input and adds a records in the data base for that specific user  $query -> bindValue(1, $firstName);  $query -> bindValue(2, $lastName);  $query -> bindValue(3, $email);  $query -> bindValue(4, $dob);  $query -> bindValue(5, $salt);  $query -> bindValue(6, $password);  $query -> execute();  header("Refresh: 1; url = home.php?");    } else {    echo "<p align=center>You have not properly filled the form</p>";  }    } else {    echo "Password aren't the same!";    }          }  ?>  <!DOCTYPE html>  <html>  <body>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id='topBanner'>  <h1> REGISTER PAGE </h1>  </div>  <div id ='backgroundImage'>  <form method='post' input align = "center">  <input type='text' name='FirstName' placeholder='FirstName' required> <br>  <input type='text' name='LastName' placeholder='LastName' required> <br>  <input type='email' name='Email' placeholder='Email' required> <br>  <input type='date' name='DateOfBirth' placeholder='DateOfBirth' required> <br>  <input type='password' name='Password' placeholder='Password' required> <br>  <input type='password' name='confirmPassword' placeholder='Confirm Password' required> <br>  <input type = "submit" name = "Register" value = "Register" class='btn'/>  <br>  <br>  </form>  <form method = "post" action = "index.php" align="center">  <input type = "submit" value="Return to Log In" class='btn'></p>  </form>  </div>  </body>  </html> |
| ***Image*** |

|  |
| --- |
| Page: search.php |
| ***Code***  <html>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>Search</h1>  </div>  </html>  <?php  /\*  Search Page should search for a routine in the database  Give user options: search by rating/muscle trained  Display Results by showing matching words  Allow routine to be shown by clicking on name  Allow user to have the option to "Add Routine" in to Time Table and/or Log  \*/  require\_once "core/init.php";  include('ColumnButtons.html');  $newSearch = new Search;  if( isset ($\_POST['submitSearch'])){  $sortVar = $\_POST['sort'];  echo "<br>";  echo "Sorted by: ".$sortVar;  echo "<br>";  }  if( isset ($\_POST['submitSearch'])){  $searchVar = $\_POST['search'];  echo "Searching for ".$searchVar;  echo "<br>";  $searchInput = $\_POST['SearchInput'];  if ($searchVar == "Name" or $searchVar == "Creator"){  $newSearch -> searchRoutine($searchInput, $sortVar, $searchVar);  }elseif ($searchVar == "Muscle"){  echo "Searching for muscle".$searchInput;  $newSearch -> searchRoutineMuscle($searchInput, $sortVar);    }    }  ?>  <html>  <table border = '1px' id='searchTable'>  <th>  <form method="post">  Sort by:  <select name="sort"">  <option value="Rating">Rating</option>  <option value="DatePerformed">DatePerformed</option>  </select>    <br>  <br>  Search By:  <select name="search"">  <option value="Name">Name</option>  <option value="Creator">Creator</option>  <option value="Muscle">Muscle</option>  </select>    <input type='text' name='SearchInput' placeholder='Search input'>    <input type='submit' name='submitSearch' class='btn' value="Search">  </form>  </th>  </table>  </html> |
| ***Image*** |

|  |
| --- |
| Page: forgotpassword.php |
| ***Code***  *<?php*  */\**  *1. User requests change password*  *2. Email sent to user, with a code. Code can be generated using a salt on their current password*  *User enters the code. If match allow user to overwrite their current password*  *\*/*  *require\_once('core/init.php');*  *if (isset($\_POST['returnIndex'])){*  *if (isset($\_SESSION['id']) or isset($\_SESSION['code'])){*  *$\_SESSION['id'] = null;*  *$\_SESSION['code']=null;*  *}*  *header("Location: index.php");*    *}*    *if (isset($\_POST['Send'])){*  *$email = $\_POST['Email'];*  *if(checkEmailExists($email)){*  *$message = generateRecoveryCode();*  *$\_SESSION['code']=$message;*  *$subjectTitle = "Forgetten Password Recovery Code";*  *$from = "GymPlanner@coursework.com";*  *$email = new EmailObj($email,'forgot');*  *$email->createBody($message);*  *echo "<form method='post'>";*  *echo "<input type='text' name='code' placeholder='enter code' required>";*  *echo "<input type = 'submit' name = 'enter' value = 'enter' class='btn'/>";*  *echo "</form>";*  *}else{*  *echo "No such email exists";*  *}*  *}*  *if (isset($\_POST['enter'])){*  *$code = $\_POST['code'];*  *echo $code;*  *echo "enter is pressed";*    *//if($code == '1111'){ // 1111 is used as a dummy code as email cannot be sent*  *if($code = $\_SESSION['code']){*  *// User is verfied without needing his password*    *echo "<form method='post'>";*  *echo "<input type='text' name='newPassword' placeholder='New Password' required>";*  *echo "<input type = 'submit' name = 'newPasswordSubmit' class='btn'/>";*  *echo "</form>";*  *}*  *}*  *if (isset($\_POST['newPasswordSubmit'])){*  *echo $\_SESSION['id'];*  *$user = new User ($\_SESSION['id']);*  *$user -> resetPassword($\_POST['newPassword'], $user ->getEmail());*  *}*  */\**  *function changePassword($email){*    *if (isset($\_POST['newPasswordSubmit'])){*  *if ($\_POST['newPassword'] == $\_POST['newPassword2']){*  *$salt = mcrypt\_create\_iv(32);*  *$password = hash("sha256", $\_POST['newPassword'], $salt);*  *$query = DB::getInstance() -> prep("UPDATE users SET Password = ? WHERE Email= ?");*  *$query -> bindValue(1, $password);*  *$query -> bindValue(2, $email);*  *$query -> execute();*  *echo "Password Successfully changed! ";*  *header("Location: Index.php");*  *}else{*  *echo "Please try again, you have inputted it incorrectly ";*  *}*  *}*  *}*  *\*/*  *function checkEmailExists($email){*  *$query = DB::getInstance()->prep("SELECT \* FROM users WHERE Email = ?");*  *$query -> bindValue(1,$email);*  *$query -> execute();*  *$results = $query->fetchAll(PDO::FETCH\_OBJ);*  *//print\_r($results);*  *if ($results){*  *$\_SESSION['id'] = $results[0]->UserId;*  *return true;*  *}else{*  *return false;*  *}*  *}*  *function generateRecoveryCode(){*  *$randomNum = random\_int(100000, 999999);*  *return $randomNum;*  *}*  *?>*  *<!DOCTYPE html>*  *<html>*  *<body>*  *<link rel='stylesheet' type='text/css' href='Style.css'>*  *<div id='topBanner'>*  *<h1> Enter Email to Send Recovery Password </h1>*  *</div>*  *<div id ='backgroundImage'>*  *<form method='post' input align = "center">*  *<input type='email' name='Email' placeholder='Email' required> <br>*  *<input type = "submit" name = "Send" value = "Send" class='btn'/>*  *<br>*  *<br>*  *</form>*  *<form method = "post" align="center">*  *<input type = "submit" value="Return to Log In" name='returnIndex' class='btn'></p>*  *</form>*  *</div>*  *</body>*  *</html>* |
| ***Image*** |

|  |
| --- |
| Page: timetable.php |
| <html>  <link rel='stylesheet' type='text/css' href='Style.css'>  <div id = "topBanner">  <h1>TimeTable</h1>  </div>  </html>  <?php  /\*  Time table will show  - Days of the Week  - The body part(s) to be trained on that day  User must be able to change timetable  \*/  require\_once "core/init.php";  include('ColumnButtons.html');  $user = new User();  if(!$user->isLoggedIn()){  header('Location: index.php');  }  $timetable = new TimeTable;  ?> |
|  |