**Student Management System Checklist**

Table of Contents

[Checklist 3](#_Toc130918534)

[Login page 5](#_Toc130918535)

[Frontend features and languages : 5](#_Toc130918536)

[Backend features and languages 5](#_Toc130918537)

[Register page 6](#_Toc130918538)

[Frontend features and languages: 6](#_Toc130918539)

[Backend features and languages: 6](#_Toc130918540)

[User Profile 8](#_Toc130918541)

[Frontend features and languages 8](#_Toc130918542)

[Backend features and languages 8](#_Toc130918543)

[Calendar Page 9](#_Toc130918544)

[Frontend features and languages: 9](#_Toc130918545)

[Backend features and languages: 10](#_Toc130918546)

[Study Tracking/Study Diary/Web Development page 11](#_Toc130918547)

[Frontend features and languages 11](#_Toc130918548)

[Backend features and languages 12](#_Toc130918549)

[Study Tracking/My Grades page 13](#_Toc130918550)

[Frontend features and languages: 13](#_Toc130918551)

[Backend features and languages: 14](#_Toc130918552)

[Study Tracking/Grade Calculator/Module Calculator page 15](#_Toc130918553)

[Frontend features and languages: 16](#_Toc130918554)

[Backend features and languages: 16](#_Toc130918555)

[Study Tracking/Grade Calculator/Year Calculator page 18](#_Toc130918556)

[Frontend features and languages: 19](#_Toc130918557)

[Backend features and languages: 19](#_Toc130918558)

[Study Tracking/Grade Calculator/Undergraduate Degree Calculator page 21](#_Toc130918559)

[Frontend features and languages: 22](#_Toc130918560)

[Backend features and languages: 22](#_Toc130918561)

[Fitness Tracking/Exercise Diary/Monday page 24](#_Toc130918562)

[Frontend features and languages: 25](#_Toc130918563)

[Backend features and languages: 26](#_Toc130918564)

[Fitness Tracking/Food Diary/Daily Nutrition Goals page 27](#_Toc130918565)

[Frontend features and languages: 27](#_Toc130918566)

[Backend features and languages: 28](#_Toc130918567)

[Fitness Tracking/Food Diary/Calories by Meals page 29](#_Toc130918568)

[Frontend languages and languages: 32](#_Toc130918569)

[Backend features and languages: 32](#_Toc130918570)

# Checklist

* Create a student management system based on study and fitness.
* MVC design pattern
* Frontend programming languages
  + HTML
  + CSS
  + JAVASCRIPT (Bootstrap framework)
* Backend Programming languages
  + PHP (Bootstrap framework)
  + MySQL (PDO)
    - There should be a database table for each page including primary key and more.
* Communication between the frontend and backend: AJAX and JSON
* Calendar - Full calendar.IO
  + Use JSON to format the data retrieved from the server and display on calendar.
* Testing the web application

1. Testing login & register
2. Testing calendar
3. Testing Student Diary
4. Testing My Grades
5. Testing Grade Calculators
6. Testing Exercise diary
7. Testing Daily Nutritional Goals
8. Testing Calories by Meals

* create a report:
  + for each page, discuss whether there were issues occurred. If there was issues occurred, then explain in detail the issue and provide the solution of how you fixed it/ or solved it.
  + Screenshot the issue and solution from the code to compare the difference.
  + Create Entity Relationship Diagram
  + A template will be provided to allow you document the work.
* One-one demo for the web application will be done to test and ensure that the web application is applicable.

# Login page

The following features for the login page includes:

* **Email validation**: Student uses the correct format to enter their email address.
* **Password validation**: Student uses the correct format to enter their password.

## Frontend features and languages :

* HTML: This is the foundation of any web page and defines the structure and content of the page.
* CSS: This is used to style the HTML elements, including layout, colours, fonts, and animations.
* JavaScript: This is used to add interactivity to the page, including validating user input and making AJAX requests to the server.
* Email validation: Use HTML input type="email" to validate the email address format and display a warning message if the format is incorrect.
* Password validation: Use HTML input type="password" to mask the password input and validate the password length and complexity using JavaScript, displaying a warning message if the password is too weak.
* AJAX requests: Use JavaScript and AJAX to submit the login form data to the server and receive a response without refreshing the page.
* Form validation: Use JavaScript to validate the login form data before submitting it to the server to ensure that it meets the expected format and constraints.

## Backend features and languages

* PHP
* MySQL
* Bootstrap
* The process of logging in:

1. Password hashing: Use PHP's password\_hash() function to hash the password with bcrypt before storing it in the database.
2. Input validation: Use PHP to validate the email and password server-side, checking for the correct format and length before attempting to authenticate the user.

# Register page

The following features for the register page includes:

* **First name validation**: Student enters their first name.
* **Last name validation**: Student enter their last name.
* **Email validation**: Student uses the correct format to enter their email address.
* **Password validation**: Student uses the correct format to enter their password.
* **Repeat password validation**: Student uses the correct format to enter their repeat password.
* **Profile picture**: Student is able to upload profile image. Student can also remove their profile image and upload again.

## Frontend features and languages:

* HTML
* CSS
* JavaScript for form validation and user interface.
* First name validation: HTML form validation using the required attribute and JavaScript validation to check for valid characters only.
* Last name validation: HTML form validation using the required attribute and JavaScript validation to check for valid characters only.
* Email validation: HTML form validation using the type="email" attribute and JavaScript validation to check for valid email format.
* Password validation: HTML form validation using the required attribute and JavaScript validation to check for valid password format.
* Repeat password validation: HTML form validation using the required attribute and JavaScript validation to check that the repeat password matches the original password.
* Profile Image validation: HTML form validation using the required attribute and JavaScript validation.

## Backend features and languages:

* PHP
* MySQL: for database management and storage of user information. When a user successfully registers, their information is stored in the MySQL database for future reference.
* Error handling: Appropriate error messages are displayed if any validation errors occur during the registration process, including if the email address is already in use.
* Server-side validation using PHP: Validate first name, last name, email, password, and repeat password fields to ensure that they meet the desired format and requirements. This prevents users from submitting incorrect or malicious data. Each validation is described in the frontend security features in detail.
* Secure password storage: Hash and salt the student’s password before storing it in the database using bcrypt.

# User Profile

Here are the features for the User Profile page:

* **Profile image**: Student can edit their profile image:
  1. Clicking on the profile image.
  2. Student can update their profile image by clicking on the ‘Choose Photo’.
  3. Student can also delete photo by clicking on the ‘Delete Photo’.
* **Full Name**: Student can edit their full name by entering their new full name in the placeholder.
* **Email Address**: Student can edit their email address by entering their new email address in the placeholder.
* **Password**: Student can edit their password by entering their new password in the placeholder.
* **Repeat Password**: Student enters new password again by entering in the placeholder.

## Frontend features and languages

* Profile image: The frontend should display the user's current profile image, and allow them to click on it to upload a new image. This can be implemented using HTML, CSS, and JavaScript, with the addition of a library or framework like React or Vue.js for smoother UI interactions.
* Full Name: The frontend should display the user's current full name and allow them to edit it by clicking on the field and entering a new value. This can be implemented using HTML and CSS, with JavaScript to handle the editing functionality.
* Email Address: The frontend should display the user's current email address and allow them to edit it by clicking on the field and entering a new value. This can be implemented using HTML and CSS, with JavaScript to handle the editing functionality.
* Password and Repeat Password: The frontend should display the password fields and allow the user to enter a new password and confirm it by entering it again. This can be implemented using HTML and CSS, with JavaScript to handle the validation and submission of the form.

## Backend features and languages

* **Profile image**: The backend should handle the upload and deletion of profile images and store the image in a database. This can be implemented using PHP (Bootstrap framework) , and MySQL as the database management system.
* **Full Name, Email Address, Password, and Repeat Password**: The backend should handle the submission of the form and update the user's information in the database. This can be implemented using PHP (bootstrap framework), and MySQL as the database management system.
* The communication between the frontend and backend can be handled using AJAX.

# Calendar Page

For a student management system based on a student's study and fitness, the calendar page includes the following features:

* **Calendar view**: Student can view the calendar by day, week, month.
* **Adding schedules**: Students can add schedule om the calendar:
  + Student clicking on the + button.
  + Student enter task name.
  + Student enter start time.
  + Student enter end time.
  + Student choose colour to represent their task.
* **Removing schedules**: Students can remove schedule on the calendar:
  + Students clicks the task on the calendar.
  + Student clicks on the red ‘Remove’ button.
  + Task is removed from the calendar.
* **Updating Schedules**: Students can update schedule on the calendar:
  + Student clicks the task on the calendar.
  + Student clicks on the green ‘Edit’ button.
  + Student can edit the ‘TITLE’, ‘STARTS’, ‘ENDS’.
  + Student can choose different colour to represent their task.
* **Tooltip**: To allows student to remove/update tasks on the calendar, a graphical UI element (tooltip) is displayed, allowing the student to click on their activity and have the ability to update and remove.
* **Day, weekly & monthly roll over**
* Student can use the day roll over feature by clicking on the next/previous arrows to enter the specific meal with the nutritional data.

## Frontend features and languages:

* HTML, CSS, and JavaScript are used to create the calendar page, including the user interface for adding, editing, and deleting events.
* The Bootstrap framework is used to create a responsive design for the calendar page, ensuring that it is easily viewable on different devices.
* Fullcalendar.io: Using JavaScript framework to create the calendar.
* JavaScript is used to add interactivity to the page, such as handling user inputs, updating the calendar view, and sending requests to the backend API.
  + When a user adds a new event, JavaScript collects the input values, validates them, and sends a POST request to the backend API to create the event.
  + When a user edits or deletes an existing event, JavaScript sends a PUT or DELETE request to the backend API, respectively.
  + The calendar view can be updated dynamically by using JavaScript to fetch calendar events from the backend API and displaying them on the page.

## Backend features and languages:

* PHP: handle data processing and manage communication with the database.
* RESTful API: This is a standard way of communicating between the front-end and the back-end, enabling data to be transmitted in a structured way.
* Database: This is where the data for the calendar will be stored and retrieved from. Popular database choices include MySQL.
* The process:

1. The backend API is built using PHP and follows the RESTful API design principles.
2. The API endpoints are defined to handle various CRUD (Create, Read, Update, Delete) operations on the calendar events.
3. When a POST request is received to create a new event, the API endpoint validates the input data, creates a new event record in the database, and returns a response to the frontend with the newly created event data.
4. When a PUT request is received to update an existing event, the API endpoint validates the input data, updates the event record in the database, and returns a response to the frontend with the updated event data.
5. When a DELETE request is received to delete an existing event, the API endpoint deletes the event record from the database and returns a response to the frontend.
6. Input data validation is performed on the backend to ensure that only valid and properly formatted data is stored in the database.
7. The MySQL database is used to store calendar events and other relevant information.

# Study Tracking/Study Diary/Web Development page

The Study Diary will allow students to add their modules (e.g., web development, business management, software management). So, the features will apply to the rest of the modules. It is essentially a ToDo list. For the web development page, the following features included:

* **Task Name**: Student enters the name of the task.
* **Task Description**: Student enters description of task.
* **Due dates**: The student should be able to add a due date for each task, which will display a small graphical calendar UI for the user to select their dates or they can manually enter the date.
* **Status**: Students enter the status of the task.
* **Task priority**: The student should be able to assign a priority level to each task. A small graphical UI shows a high, medium, or low priority level to allow the student to select the level of priority.
* **Adding new tasks**: The student is able to add new task by clicking on the ‘Add Task’ button.
* **Removing tasks**: The student is able to remove the task by clicking on the remove button.
* **Completed tasks**: The student should be able to mark a task as completed once it is finished by clicking on the tick box on the left hand side.
* **Add module**: The student can add another module by clicking the add button.
* **Save data**: The student save their input data by clicking on the save button.
* **Update data**: The student can return back to the page and update the necessary data and save the data.

## Frontend features and languages

* HTML: This is the foundation of any web page and defines the structure and content of the page. Use HTML to create a simple form with input fields for the task description and due date.

1. A form is created with input fields for task description and due date.
2. The input field for task description has a "required" attribute, ensuring that the user provides a value.
3. The input field for due date has a "type" attribute set to "date," which provides a date picker and ensures valid date format.

* CSS: This is used to style the HTML elements, including layout, colours, fonts, and animations.
* JavaScript: This is used to add interactivity to the page, including adding and removing tasks, prioritizing tasks, and filtering tasks. JavaScript to validate user inputs such as task descriptions and due dates, ensuring they meet the required format and constraints before submitting to the server:
  1. A function is created to handle the form submission event.
  2. When the form is submitted, the function checks if the task description and due date input fields meet the required constraints (e.g., minimum length for task description).
  3. If the constraints are not met, the function displays an error message, prevents the form from being submitted, and requests the user to correct the input.
  4. If the input fields meet the constraints, the function allows the form to be submitted to the server for further processing.

Backend features and languages

* PHP: To handle data processing and manage communication with the database. Validate user inputs on the server-side using PHP to ensure only valid data is accepted and processed, even if frontend validation is bypassed:

1. The server receives a POST request with task description and due date data.
2. PHP code checks if the request method is POST.
3. PHP code retrieves the task description and due date data from the form data.
4. PHP code initializes an $errors array to store any validation errors.
5. PHP code validates the task description by checking that it is not empty and has a minimum length of 3 characters.
6. PHP code validates the due date by checking that it is not empty and matches the required format using a regular expression.
7. If there are no errors, the PHP code processes the validated data (e.g., saves it to the database).
8. If there are errors, the PHP code displays them to the user by outputting each error message in a loop.
9. The user is notified of any validation errors and can correct the input accordingly before resubmitting the form.

* MySQL: This is where the data for the to-do list will be stored and retrieved from.

# Study Tracking/My Gradespage

The ‘My Grades’ page will allow students to add their assignment marks (e.g., web development, business management, software management). It is essentially a ToDo list. For the My Grades’ page, the following features included:

* **Module name**: Student enter module name. E.g., Web Development
* **Assignment name**: Student enters assignment name. E.g., Web Dev 1
* **Assignment marks**: Student enters assignment marks. E.g., 70
* **Add assignment/exam** : The student should be able to add another assignment.
* **Remove assignment/exam**: The student should be able remove the assignment.
* **Add new module**: The student can add a new module and enter the assignment name and assignment marks (%).
* **Add new section**: The student can add new section and enter the name. E.g., Year3, Year 4
* **Delete section**: The student can delete section by clicking on the name and clicking the delete button.
* **Edit section**: The student can edit the name of the section by clicking on the name and clicking the edit name button.
* **Save data**: The student save their input data by clicking on the save button.
* **Update data**: The student can return back to the page and update the necessary data and save the data.

## Frontend features and languages:

* HTML: This is the foundation of any web page and defines the structure and content of the page.
* CSS: This is used to style the HTML elements, including layout, colours, fonts, and animations.
* JavaScript: This is used to add interactivity to the page, including adding and removing tasks, prioritizing tasks, and filtering tasks.
* Input validation: Use JavaScript to validate user inputs such as assignment names and marks before submitting the form. Ensure that they meet the required format and constraints.
  + Use of JavaScript for validating user inputs for assignment names and marks, along with accompanying HTML form:
  1. The HTML file contains a form with two input fields: assignment name and assignment marks, and a submit button. There's also a paragraph element for displaying error messages.
  2. The JavaScript file, grades-validation.js, is linked to the HTML file and contains the validation logic for the form.
  3. When the form is submitted, the JavaScript code adds an event listener to the form, which prevents the default form submission behavior using event.preventDefault().
  4. The JavaScript code retrieves the values entered by the user in the assignment name and assignment marks input fields.
  5. It then checks if the assignment name is at least 3 characters long. If not, it displays an error message in the "error-message" paragraph element and stops further execution.
  6. Next, it checks if the assignment marks are within the range of 0 to 100. If not, it displays an error message in the "error-message" paragraph element and stops further execution.
  7. If both the assignment name and marks pass the validation checks, any previous error message is cleared.
  8. Finally, with the validation passed, you can submit the form or send the data to the server using AJAX. In this example, the valid data is logged to the console as a demonstration.

## Backend features and languages:

* PHP: To handle tasks such as form validation, adding and removing grades, saving grades, and integrating with the My Grades page.
* Database: MySQL to store and retrieve grade data.
* Server-side validation: Validate user inputs with PHP, even if frontend validation is bypassed. Ensure that the inputs meet the required format and constraints.
  + The use of PHP validating user inputs such as an assignment name and marks, even if the frontend is bypassed:
  1. The server checks if the request method is POST.
  2. The assignment name and assignment marks are retrieved from the submitted form data.
  3. An empty array called $errors is created to store any validation errors.
  4. The assignment name is checked for minimum length requirements (at least 3 characters). If it's too short, an error message is added to the $errors array.
  5. The assignment marks are checked to ensure they are a number between 0 and 100. If they don't meet the requirements, an error message is added to the $errors array.
  6. If there are no errors in the $errors array, the data is processed (e.g., saved to the database).
  7. If there are errors, they are displayed to the user using a foreach loop.

# Study Tracking/Grade Calculator/Module Calculator page

The Module calculator can be used to generated weighted grades for a module consisting of assignments, e.g. an exam and coursework. Here are features added to the Module Calculator:

* **Assignment name**: Student enters assignment name. E.g., Information Security Management
* **Assignment grade (%)**: Student enter assignment grade. E.g., 87
* **Assignment Weighting (%)**: Student enters assignment’s weighting . E.g., 50)
* **Calculation for Assignment Weighted (%)**: The calculation is done by firstly multiplying the grade by its weighting to generate a weighted grade for that row. The weighted grades are then summed to generate an overall weighted grade for the module. Then finally, the weighted grades are then summed to create the overall result for the module. https://www.gradecalc.co.uk/uni/module - look at ‘How does it work?’ section for the calculation.
* **Formulae for Assignment Weighted (%) Calculation**: The formula used to calculate the overall weighted grade is typically:
  + ((grade1 \* weighting1) + (grade2 \* weighting2) + ... + (gradeN \* weightingN)) / (total weighting)
  + Where N is the number of components and total weighting is the sum of all the weightings.
* **Display of overall degree classification**: After the calculation is completed, the calculator could display the student's predicted degree classification (e.g. First Class Honours, Second Class Honours, etc.):
  + **F**irst class (1st) is equivalent to => 70% .
  + Higher Second class (2:1) is equivalent to 60% => 69.99%.
  + Lower second class (2:2) is equivalent to 50% => 59.99%
  + Third class (3rd) is equivalent to 40% => 49.99%
  + Fail is equivalent to 0% => 39.99%
* **Add Module**: Student is able to add module by clicking on the ‘Add another row’.
* **Remove Module** : Student is able to remove module by clicking on the remove icon.
* **Refresh page**: Once student moves onto the next page, refresh the calculator to the next page. It is similar like a calculator where you enter a new calculation.

## Frontend features and languages:

* HTML
* CSS
* JavaScript: for creating the user interface and handling user input and calculations.
* Input validation: Use JavaScript to validate user inputs, such as ensuring the input values are within the expected range and format. For example, checking that the grade and weighting values are between 0 and 100.
  + The use of JavaScript to validate user input for the grade and weighting values in a module calculator:
  1. The code creates a function named validateInput() that is called when the user clicks the submit button on the form.
  2. The function gets the values for each row in the table by selecting the table rows and accessing their respective input fields.
  3. The function then loops through each row and validates the grade and weighting values.
  4. For each row, the function checks if the grade and weighting values are within the expected range of 0 to 100. If not, an error message is displayed, and the function returns false.
  5. If all input values pass validation, the function returns true, allowing the form to be submitted to the server for further processing.
  6. By validating user input on the client-side, this approach helps ensure that only valid data is submitted to the server, reducing the risk of unexpected behaviour or security vulnerabilities.

## Backend features and languages:

* PHP: for handling server-side requests and responses
* The page performs the calculation in real-time and displays the result to the user based on the input provided. Therefore, the data is not persistent and there is no need to store it in a database table.
* Server-side validation: Validate user inputs with PHP to ensure that they meet the required format and constraints, even if frontend validation is bypassed.
  1. Validating user inputs with PHP for the Module Calculator:
  2. The PHP script receives user inputs for the module calculator, such as the name, grade, and weighting for each component.
  3. The PHP script validates the user inputs, ensuring that they meet the required format and constraints, such as being within the range of 0 to 100 for the grade and weighting.
  4. The PHP script calculates the weighted grade for each component, multiplying the grade by the weighting and dividing by 100.
  5. The PHP script sums the weighted grades to generate an overall weighted grade for the module.
  6. The PHP script determines the degree classification based on the overall weighted grade using the updated version of displaying the student's overall degree.
  7. The PHP script saves the user inputs and the calculated results to the database.
  8. The PHP script sends a response to the client-side, including the calculated overall weighted grade and predicted degree classification.
  9. The client-side JavaScript displays the results on the page, showing the user their predicted degree classification for the module.

# Study Tracking/Grade Calculator/Year Calculator page

This calculator can be used to generated weighted grades for a year consisting of credit based modules (accumulating a figure of 120 credits generally).

Here are features added to the Year Calculator page include:

* **Module name**: Student enter module name.
* **Module grade(%)**: Student enters the overall grade for the module.
* **Module credit**: Student enters the module. It should accumulate to a figure of 120 credits generally.
* **Calculation for module weighted (%)** : The calculation is done by firstly adding the total credits provided and turns the individual credits into percentages of the total credits supplied. The weighted grades are then summed to generate an overall weighted grade. We then multiply the credit by 100 and divide by the total credits. We then multiply the grade by the weighted credit and divide by 100 for the result. We'd then complete the same process for the other module.

https://www.gradecalc.co.uk/uni/year - look at ‘How does it work?’.

* **Formula for calculating the overall weighted grade for the year**:
  + Overall Weighted Grade = (Sum of Weighted Grades for All Modules) / Total Credits

Where:

* + Sum of Weighted Grades for All Modules: The sum of each module's weighted grade, which is calculated as follows:
    - Weighted Grade = (Grade x Weighted Credit) / 100
    - Grade: The grade earned for the module (a percentage between 0 and 100)
    - Weighted Credit: The credit value for the module multiplied by its weighting (a percentage between 0 and 100)
  + Total Credits: The total number of credits for all modules in the year (usually 120 for a full-time student)
* **Display of overall degree classification**: After the calculation is completed, the calculator could display the student's predicted degree classification (e.g. First Class Honours, Second Class Honours, etc.):
  + **F**irst class (1st) is equivalent to => 70% .
  + Higher Second class (2:1) is equivalent to 60% => 69.99%.
  + Lower second class (2:2) is equivalent to 50% => 59.99%
  + Third class (3rd) is equivalent to 40% => 49.99%
  + Fail is equivalent to 0% => 39.99%
* **Add new row**: The student is able to add new module by clicking on the add button.
* **Refresh page**: Once student moves onto the next page, refresh the calculator to the next page. It is similar like a calculator where you enter a new calculation.

## Frontend features and languages:

* HTML
* CSS
* JavaScript: creating the user interface and handling user interactions.
* Input validation: Use JavaScript to validate user inputs such as module name, grade, credit, and weighted values to ensure they meet the required format and constraints before submitting the form.
  1. The use of JavaScript to validate user inputs:
  2. The validateForm function is called when the form is submitted.
  3. The function retrieves the values from the input fields: moduleName, grade, credit, and weighted.
  4. It checks if the moduleName is not empty.
  5. The function validates the grade input to ensure it is a number between 0 and 100.
  6. It validates the credit input, making sure it is a number between 0 and 120.
  7. The function checks the weighted input to ensure it is a number between 0 and 100.
  8. If any validation fails, an alert message is displayed, and the form submission is prevented by returning false.
  9. If all validations pass, the function returns true, allowing the form submission to proceed.

## Backend features and languages:

* PHP or any server-side scripting language.
* MySQL handle the storage and retrieval of data.
* Server-side validation: Use PHP or another server-side language to validate user inputs even if frontend validation is bypassed. Ensure that the inputs meet the required format and constraints.
  1. Steps to validate user inputs to the module name, grade, credit and weighted value using PHP:
  2. We check if the form was submitted using the POST method.
  3. We retrieve user inputs from the $\_POST array.
  4. We initialize an empty array $errors to store validation error messages.
  5. Validate the module name:
     1. Check if the module name is empty or shorter than 3 characters.
     2. If it is, add an error message to the $errors array.
  6. Validate the grade:

1. Check if the grade is a numeric value between 0 and 100 (inclusive).
2. If it is not, add an error message to the $errors array.
   1. Validate the credit:
3. Check if the credit is a numeric value between 0 and 120 (inclusive).
4. If it is not, add an error message to the $errors array.
   1. Validate the weighted value:
5. Check if the weighted value is a numeric value between 0 and 100 (inclusive).
6. If it is not, add an error message to the $errors array.
   1. After the validation checks, we determine if there are any errors:
7. If the $errors array is empty (no errors), we process the validated data (e.g., save it to the database).
8. If there are errors, we display the error messages to the user using a loop to iterate through the $errors array.

# Study Tracking/Grade Calculator/Undergraduate Degree Calculator page

Here are the features added to the Undergraduate Degree Calculator page are:

* **Undergraduate degree name**: Student enter the undergraduate degree name. E.g., Year 2
* **Undergraduate degree grade (%)**: Student enter the undergraduate degree grade (%). E.g., 60
* **Undergraduate degree weighting (%)**: Student enter the undergraduate degree weighting (%). E.g., 25
* **Calculation for undergraduate degree weighted (%)**: The calculation is done:
  1. First, it multiplies the grade by its weighting to generate a weighted grade for that year.
  2. The weighted grades are then summed to generate an overall weighted grade for the degree.
  3. The weighted grades are then summed to create the overall result for the degree.
  + Calculation: The calculation starts by summing up the total credits entered by the student. We then loop through each row of module data, and for each module, we calculate the percentage of the total credits the module accounts for. We then calculate the weighted grade for the module by multiplying the grade by the weighted percentage (the percentage of the total credits the module accounts for) and dividing by 100. We then multiply the credit by 100 and divide by the total credits to get the percentage of the year that the module is worth. Finally, we sum the weighted grades to get the overall weighted grade for the year. The overall degree classification is determined based on the overall weighted grade using the same criteria as outlined earlier.
* **Formula for undergraduate degree calculation:**

Overall Weighted Grade = [((Grade1 \* Weighting1)/100) + ((Grade2 \* Weighting2)/100) + ... + ((GradeN \* WeightingN)/100)]

Where:

* + Grade1, Grade2, ..., GradeN are the grades for each year of the degree
  + Weighting1, Weighting2, ..., WeightingN are the weightings for each year of the degree
* **Display of overall degree classification**: After the calculation is completed, the calculator could display the student's predicted degree classification (e.g. First Class Honours, Second Class Honours, etc.):
  + First class (1st) is equivalent to => 70% .
  + Higher Second class (2:1) is equivalent to 60% => 69.99%.
  + Lower second class (2:2) is equivalent to 50% => 59.99%
  + Third class (3rd) is equivalent to 40% => 49.99%
  + Fail is equivalent to 0% => 39.99%
* https://www.gradecalc.co.uk/uni/undergraduate-degree - look at ‘How does it work?’ section for calculation.
* **Refresh page**: Once student moves onto the next page, refresh the calculator to the next page. It is similar like a calculator where you enter a new calculation.

## Frontend features and languages:

* HTML
* CSS
* JavaScript for creating the user interface and visualisations.
* Input validation: Use JavaScript to validate user inputs, such as module name, grade, and weighting, ensuring they meet the required format and constraints before submitting the form.

1. Define a JavaScript function called validateForm to validate user inputs.
2. Retrieve user inputs for module name, grade, and weighting from the form.
3. Initialize an empty errorMessage string to store any validation error messages.
4. Check if the module name is not empty; if it is, add an error message to errorMessage.
5. Check if the grade is a number between 0 and 100; if not, add an error message to errorMessage.
6. Check if the weighting is a number between 0 and 100; if not, add an error message to errorMessage.
7. If errorMessage is not empty, display it and prevent form submission by returning false.
8. If no validation errors, allow form submission by returning true.
9. Attach the validateForm function to the form's onsubmit event to trigger validation on submission.

## Backend features and languages:

* PHP for server-side processing
* MySQL for storing and retrieving data.
* Input validation: Use PHP to validate user inputs even if frontend validation is bypassed. Ensure that the inputs meet the required format and constraints.
* Undergraduate Degree Calculation: After user validation, calculation is performed to find the student’s undergraduate degree:
  1. Define a function calculate\_weighted\_grade that takes two arrays as input: grades and weightings.
  2. Inside the function, initialize a variable weighted\_grades\_sum to store the sum of the weighted grades.
  3. Loop through the grades array, and for each grade, multiply it by its corresponding weighting, then add the result to weighted\_grades\_sum.
  4. Return weighted\_grades\_sum from the function.
  5. Define another function get\_degree\_classification that takes the overall weighted grade as input.
  6. Inside the function, use conditional statements to determine and return the degree classification based on the overall weighted grade.
  7. After validating user inputs, create arrays for grades and weightings with the validated values.
  8. Call calculate\_weighted\_grade with the grades and weightings arrays to get the overall weighted grade.
  9. Call get\_degree\_classification with the overall weighted grade to determine the degree classification.
  10. Save or display the overall weighted grade and degree classification as needed.
* Regarding the use of PHP for server-side validation, it's worth noting that PHP provides a built-in function, filter\_var(), that can be used to validate input values. For instance, you can use filter\_var() with the FILTER\_VALIDATE\_FLOAT filter to validate numeric values, and with the FILTER\_VALIDATE\_REGEXP filter to validate values that match a specific pattern. Using filter\_var() can simplify your validation logic and reduce the risk of security vulnerabilities caused by incorrect input validation.

# Fitness Tracking/Exercise Diary/Monday page

The exercise diary includes all they day starting from Monday to Sunday, allowing the student to enter their exercises. So, most of the features will be the same on the rest of the pages. It is similar to a ToDo list. Here are features included in a fitness tracking page for a student management system with separate todo lists for strength training and cardio:

* **Fitness name input**: The students enters the name (fitness name) of the their fitness activities, the name can be anything and it can reflect on the exercises the student does. E.g., Push
* **Todo list generator for strength training**: A feature that generates separate todo lists for strength training including:
  + **Name of exercise**: Student enter the name of the exercise. E.g., Bench Press
  + **Reps**: Student enters the number reps they will perform during the exercise. E.g., 10-12 reps
  + **Sets**: Student enters the number of sets during the exercise. E.g., 3
  + **Recovery period**: Student enters the number of seconds/minutes for their rest period. E.g., 1 minute
    - A graphical UI should allow student to select minutes or seconds.
  + At the end of the strength training todo list, the student will manually enter total calories. E.g., 425 KCAL
* **Todo list generator for cardio**: A feature that generates separate todo lists for cardio including:
  + **Name of exercise**: Student enters the name of the exercise. E.g., Elliptical
  + **Duration**: Student enters the duration of the exercise. E.g., 30
  + At the end of the cardio todo list, the student will manually enter total calories. E.g., 350 KCAL
* **Add exercise**: Student is able to add exercise from the strength training & cardio todo list generator by clicking on the add button.
* **Remove exercise**: Student is able to remove exercise from the strength training & cardio todo list generator by clicking on the remove button.
* **Calories Burned**: The student responsibility is to enter their calories burned for their strength training and cardio. If the student has entered their name of the exercise, but has not entered the burned calories, this indicates the student has not completed the exercise, else there was no exercise schedule today. It includes:
  + **Strength Training Calories Burned**: The student will manually enter total calories. E.g., 425 KCAL
  + **Cardio Calories Burned**: The student will manually enter total calories. E.g., 350 KCAL
* **Total calories burned**: The calculation for ‘Total Calories Burned’ is performed by:
  + Strength Calories Burned + Cardio Calories Burned
* **Burned calories target integration**: The ‘Burned Calories Target’ data is extracted from the Fitness Tracking/Food Diary/Daily Nutritional Goal and extracted Fitness Tracking/Exercise Diary/ Monday page.
* **Remaining burned calories**: The calculation for ‘Remaining Burned Calories’ is performed by:
  + Burned Calories Target – Total Calories Burned
* **Save data**: The student save their data by clicking on the save button.
* **Update data**: The student can return back to the page and update the necessary data and save the data.

## Frontend features and languages:

* HTML: for creating the structure of the fitness tracking page.
* CSS: for styling the page and making it visually appealing.
* JavaScript: for adding interactivity to the page, such as pop-ups, sliders, and dropdown menus.
* Input validation: Use JavaScript to validate user inputs, such as ensuring the input values are within the expected range and format. For example, checking that the number of reps and sets are positive integers, and that the duration for cardio exercises is in minutes.

1. This code defines two functions validateStrengthTraining and validateCardio that retrieve the values of input fields for strength training and cardio exercises respectively.
2. It then checks if the values meet the required format and constraints, such as
   1. checking if the number of reps and sets are positive integers,
   2. if the duration for cardio exercises is in minutes.
   3. If any validation fails, it alerts the user with an appropriate message and returns false.
3. If all validations pass, it returns true.
4. These functions can be called on form submission to ensure that user inputs are valid.

* Displaying current date: Use JavaScript and Bootstrap framework to design and display current date:
  1. Create a div element in the HTML file where the date will be displayed.
  2. In the JavaScript file, use the Date() function to retrieve the current date and time.
  3. Use JavaScript's built-in methods to extract the day, date, month, and year from the current date object.
  4. Format the extracted date information into the desired format of "Monday DD MMMM YYYY".
  5. Update the innerHTML property of the div element created earlier with the formatted date string.
  6. Use Bootstrap classes and styling to format and position the div element on the page.
  7. We then display the formatted date on the page using the ‘*textContent’* property of a DOM element.

## Backend features and languages:

* PHP: for processing user data and generating dynamic content based on user input.
* MySQL : for storing and managing user data such as their exercises.

# Fitness Tracking/Food Diary/Daily Nutrition Goals page

The ‘Daily Nutritional’ page allows a student to enter their nutritional goals. Here are the features included:

* **Macro Tracker** : Student enters their **Macros** including:
  1. Student enters their Carbs
  2. Student enters their Protein
  3. Student enters their Fats
* **Consumed Calories Tracker**: Students enters their **Consumed Calories Target** for their nutrition goals.
* **Burned Calories Tracker**: Students enters their **Burned Calories Target** for their fitness goals.
* **Weight tracking**: Students enters specific data including**:** 
  1. Student enters their Starting Weight
  2. Student enters their Current Weight
  3. Student enters their Goal Weight
* **Save data**: The student save their input data by clicking on the save button.
* **Update data**: The student can return back to the page and update the necessary data and save the data.

## Frontend features and languages:

* HTML: for creating the structure of the fitness tracking page.
* CSS: for styling the page and making it visually appealing.
* JavaScript: for adding interactivity to the page, such as pop-ups, sliders, and dropdown menus.
* Input validation: Use JavaScript to validate user inputs such as Macros, Consumed Calories Target, Burned Calories Target, Starting Weight, Current Weight, and Goal Weight before submitting the form. Ensure that they meet the required format and constraints.
  + Define expected format and constraints for user inputs such as macros, consumed calories, burned calories, and weight fields.
  + Use JavaScript to validate user inputs before submission.
  + Ensure that the input values are within the expected range and format:
    - For Macros, ensure that carbs, protein, and fat are all positive integers and measured in grams.
    - For Consumed Calories Target and Burned Calories Target, ensure that they are positive integers and measured in calories (KCAL).
    - For Starting Weight, Current Weight, and Goal Weight, ensure that they are positive numbers and measured in grams.
  + Provide error messages to inform the user if the input value does not meet the required format or constraints.
  + Use regular expressions or built-in JavaScript functions to validate the input values.
  + If the values meet the requirements, allow the form submission, and save the data to the database.

## Backend features and languages:

* PHP: for processing user data and generating dynamic content based on user input.
* MySQL : for storing and managing user data such as storing, updating, removing the Macros, Consumed Calories Target, Burned Calories Target, Starting Weight, Current Weight & Goal Weight.
* Server-side validation: Validate user inputs with PHP, even if frontend validation is bypassed. Ensure that the inputs meet the required format and constraints.
  + Step to validating user inputs:
  1. Define PHP functions to validate each input field, such as validateMacros(), validateConsumedCalories(), validateBurnedCalories(), validateStartingWeight(), validateCurrentWeight(), and validateGoalWeight().
  2. Inside each function, use regular expressions or built-in PHP functions to ensure that the input values meet the required format and constraints, such as ensuring that the input is a positive integer or positive number.
  3. If the input value does not meet the required format or constraint, return an error message.
  4. In the PHP script that processes the form data, call the appropriate validation function for each input field and check if any errors were returned.
  5. If any errors were returned, display them to the user and prevent the form from being submitted to the database.
  6. If no errors were returned, proceed to save the data to the database.

# Fitness Tracking/Food Diary/Calories by Meals page

The ‘Calories by Meals’ page allows the student to enter their nutritional data. It is similar to a ToDo list. Here are the features needed for the Calories by Meal page are:

* **Todo list generator for breakfast meal input**: A feature that generates a todo lists for breakfast meal including:
  + Student enter the breakfast name.
  + Student enter breakfast serving size.
  + Student enter breakfast calories.
  + Student enter breakfast carbs.
  + Student enter breakfast protein.
  + Student enter breakfast fats.
* **Todo list generator for lunch meal input**: A feature that generates a todo lists for lunch meal including:
  1. Student enter the lunch name.
  2. Student enter lunch serving size.
  3. Student enter lunch calories.
  4. Student enter lunch carbs.
  5. Student enter lunch protein.
  6. Student enter lunch fats.
* **Todo list generator for dinner meal input**: A feature that generates a todo lists for dinner meal including:

1. Student enter the dinner name.
2. Student enter dinner serving size.
3. Student enter dinner calories.
4. Student enter dinner carbs.
5. Student enter dinner protein.
6. Student enter dinner fats.

* **Todo list generator for snacks meal input**: A feature that generates a todo lists for snacks meal including:

1. Student enter the snacks name.
2. Student enter snacks serving size.
3. Student enter snacks calories.
4. Student enter snacks carbs.
5. Student enter snacks protein.
6. Student enter snacks fats.

* **Calculation for Total Consumed per mea**l: Calculation for each meal’s total consumed is performed by:
  + **Breakfast Total Consumption**:
  1. Add all the Breakfast Total Calories
  2. Add all the Breakfast Total Carbs
  3. Add all the Breakfast Total Protein
  4. Add all the Breakfast Total Fats
  + **Lunch Total Consumption**:

1. Add all the Lunch Total Calories
2. Add all the Lunch Total Carbs
3. Add all the Lunch Total Protein
4. Add all the Lunch Total Fats
   * **Dinner Total Consumption**:
5. Add all the Dinner Total Calories
6. Add all the Dinner Total Carbs
7. Add all the Dinner Total Protein
8. Add all the Dinner Total Fats
   * **Snacks Total Consumption**:
9. Add all the Snacks Total Calories
10. Add all the Snacks Total Carbs
11. Add all the Snacks Total Protein
12. Add all the Snacks Total Fats

* **Calculation for Total Consumed**: Calculation for ‘Total Consumed’ is performed by:
  + **Total Consumed Calories**: Breakfast Total Calories + Lunch Total Calories + Dinner Total Calories + Snacks Total Calories
  + **Total Consumed Carbs**: Breakfast Total Carbs + Lunch Total Carbs + Dinner Total Carbs + Snacks Total Carbs
  + **Total Consumed Protein**: Breakfast Total Protein + Lunch Total Protein + Dinner Total Protein + Snacks Total Protein
  + **Total Consumed Fats**: Breakfast Total Fats + Lunch Total Fats + Dinner Total Fats + Snacks Total Fats
* **Consumed Calories Target integration** : The ‘Consumed Calories Target’ data and ‘Macros’ (‘CARBS’, ‘PROTEIN’ & ‘FATS’) data is extracted from the Fitness Tracking/Food Diary/Daily Nutritional Goals page and inserted into the Fitness Tracking/Food Diary/Daily Nutritional Goals page. This will show the difference between the ‘Total Consumed’ data and daily nutritional goals data.
* **Calculation for Remaining calories, carbs, protein & fats** : Calculates the’ Remaining’ is performed by:
  + **Calories**: ‘Daily Targets’ subtract the ‘Total Consumed’. The remaining calories must be updated all times, regardless of whether the student completes/incomplete entering their nutritional data.
  + **Carbs**: ‘Daily Targets’ subtract ‘Total Consumed’. The remaining carbs must be updated all times, regardless of whether the student completes/incomplete entering their nutritional data.
  + **Protein**: ‘Daily Targets’ subtract ‘Total Consumed’. The remaining protein must be updated all times, regardless of whether the student completes/incomplete entering their nutritional data.
  + **Fats**: ‘Daily Targets’ subtract ‘Total Consumed’. The remaining fats must be updated all times, regardless of whether the student completes/incomplete entering their nutritional data.
  + The remaining calories, carbs, protein, and fats can be displayed as positive/negative number.
  + The serving size data does not need to be calculated.
* **Add food:** The student can add food including:
  + Student can add food for their breakfast meal by clicking on the add button.
  + Student can add food for their lunch meal by clicking on the add button.
  + Student can add food for their dinner meal by clicking on the add button.
  + Student can add food for their snacks meal by clicking on the add button.
* **Remove food: The student can remove food including:** 
  + Student can remove food for their breakfast meal by clicking on the remove button.
  + Student can remove for their lunch meal by clicking on the remove button.
  + Student can remove food for their dinner meal by clicking on the remove button.
  + Student can remove food for their snacks meal by clicking on the remove button.
* **Save data**: The student save their input data by clicking on the save button.
* **Update data**: The student can return back to the page and update the necessary data and save the data.

## Frontend languages and languages:

* HTML: for creating the structure of the fitness tracking page.
* CSS: for styling the page and making it visually appealing.
* JavaScript: for adding interactivity to the page, such as pop-ups, sliders, and dropdown menus.
* Input validation using JavaScript to ensure that the input values meet the required format and constraints.
  + Steps for input validation using JavaScript:
  1. This function takes in the input values from the user and checks if they meet the required format and constraints.
  2. Specifically, it checks if the serving size, calories, carbs, protein, and fats are all positive numbers.
  3. If any of the input values are invalid, an alert is displayed to the user and the function returns false, indicating that the input is invalid.
  4. If all the input values are valid, the function returns true, indicating that the input is valid and can be submitted.

## Backend features and languages:

* PHP: for processing user data and generating dynamic content based on user input.
* MySQL : for storing and managing user data such as storing, updating, removing the Macros, Consumed Calories Target, Burned Calories Target, Starting Weight, Current Weight & Goal Weight.
* Sanitization and validation of user inputs on the server-side using PHP, even if frontend validation is bypassed.
* Steps using PHP to perform the three calculations:
  1. The input values are retrieved from the HTML form using the $\_POST method in PHP.
  2. The total nutrients per meal are calculated using the input values for each meal.
  3. The total nutrients for all meals are calculated by adding up the total nutrients for each meal.
  4. The consumed calories target is retrieved from the database using a PHP function.
  5. The remaining calories and remaining nutrients are calculated by subtracting the consumed nutrients from the consumed target nutrients.
  6. The calculated values are displayed in the HTML form using PHP echo statements.

Here is example:

<?php

// retrieve the input values from the form

$breakfast\_calories = $\_POST['breakfast\_calories'];

$breakfast\_carbs = $\_POST['breakfast\_carbs'];

$breakfast\_protein = $\_POST['breakfast\_protein'];

$breakfast\_fat = $\_POST['breakfast\_fat'];

$lunch\_calories = $\_POST['lunch\_calories'];

$lunch\_carbs = $\_POST['lunch\_carbs'];

$lunch\_protein = $\_POST['lunch\_protein'];

$lunch\_fat = $\_POST['lunch\_fat'];

$dinner\_calories = $\_POST['dinner\_calories'];

$dinner\_carbs = $\_POST['dinner\_carbs'];

$dinner\_protein = $\_POST['dinner\_protein'];

$dinner\_fat = $\_POST['dinner\_fat'];

// calculate the total nutrients per meal

$breakfast\_total = $breakfast\_calories + $breakfast\_carbs + $breakfast\_protein + $breakfast\_fat;

$lunch\_total = $lunch\_calories + $lunch\_carbs + $lunch\_protein + $lunch\_fat;

$dinner\_total = $dinner\_calories + $dinner\_carbs + $dinner\_protein + $dinner\_fat;

// calculate the total nutrients for all meals

$total\_calories = $breakfast\_calories + $lunch\_calories + $dinner\_calories;

$total\_carbs = $breakfast\_carbs + $lunch\_carbs + $dinner\_carbs;

$total\_protein = $breakfast\_protein + $lunch\_protein + $dinner\_protein;

$total\_fat = $breakfast\_fat + $lunch\_fat + $dinner\_fat;

// calculate the remaining calories

$consumed\_calories = $total\_calories;

$consumed\_carbs = $total\_carbs;

$consumed\_protein = $total\_protein;

$consumed\_fat = $total\_fat;

// get the consumed calories target from the database

$consumed\_calories\_target = get\_consumed\_calories\_target\_from\_db();

// calculate the remaining calories

$remaining\_calories = $consumed\_calories\_target - $consumed\_calories;

$remaining\_carbs = $consumed\_carbs - get\_carbs\_target\_from\_db();

$remaining\_fat = $consumed\_fat - get\_fat\_target\_from\_db();

?>

<!-- HTML form to display the results -->

<form>

<!-- display the total nutrients per meal -->

<h3>Total Nutrients per Meal</h3>

<p>Breakfast: <?php echo $breakfast\_total; ?></p>

<p>Lunch: <?php echo $lunch\_total; ?></p>

<p>Dinner: <?php echo $dinner\_total; ?></p>

<!-- display the total nutrients for all meals -->

<h3>Total Nutrients for All Meals</h3>

<p>Total Calories: <?php echo $total\_calories; ?></p>

<p>Total Carbs: <?php echo $total\_carbs; ?></p>

<p>Total Protein: <?php echo $total\_protein; ?></p>

<p>Total Fat: <?php echo $total\_fat; ?></p>

<!-- display the remaining calories -->

<h3>Remaining Calories</h3>

<p>Remaining Calories: <?php echo $remaining\_calories; ?></p>

<p>Remaining Carbs: <?php echo $remaining\_carbs; ?></p>

<p>Remaining Fat: <?php echo