CSE 340 PHP Motors
Final Project

Throughout the semester you have built the PHP Motors site and added CRUD functionality for vehicles and clients, have used the Model-View-Control (MVC) architecture and paid close attention to validating incoming data and using PHP to make all this work. The project below provides an opportunity to make a few "tweaks" to what you have already done and add one final piece of functionality to enhance the content of the site using these same concepts.

Video Overview

Refer to the video overview in the course materials.

Be Mindful

This is the **final project** of the course. It is meant to be challenging but do-able. As with all other activities and enhancements you are encouraged work with your learning teams to accomplish it, but all work must be your own! It will take time, plan on it - you have approximately two weeks to finish it.

Project Tasks

The final task is to build a "vehicle review" application that will allow vehicle reviews to be added to the site. The specifics are:

Create the Database Table

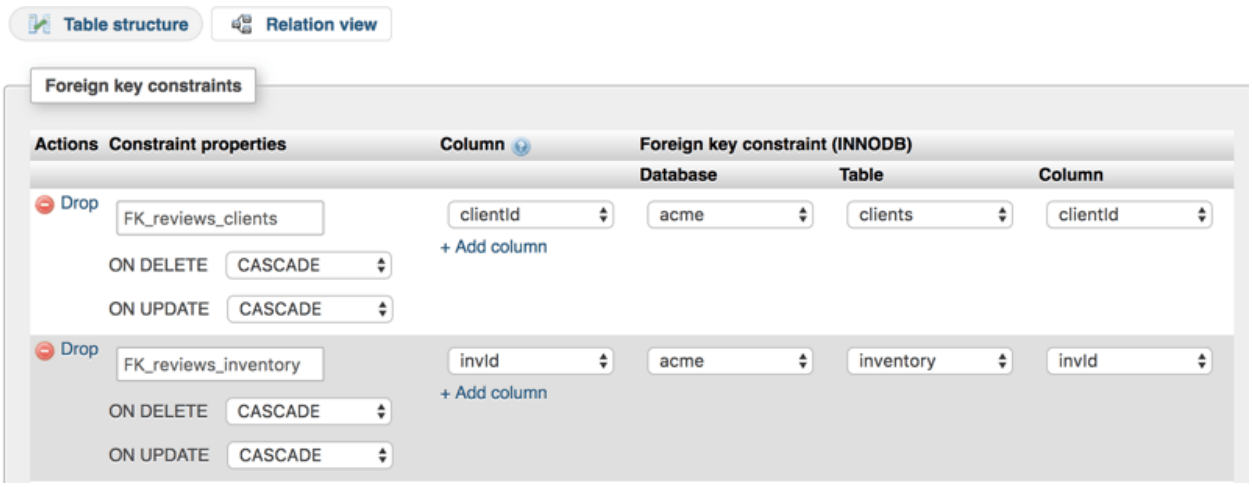
1. Build the *reviews* table in the phpmotors database following this data dictionary specification:

Name	Type	Collation	Attributes	Null	Default	Comments	Extra
reviewId	int(10)		UNSIGNED	No	*None*		AUTO_INCREMENT
reviewText	text	latin1_swedish_ci		No	*None*		
reviewDate	timestamp			No	CURRENT_TIMESTAMP		
invId	int(10)		UNSIGNED	No	*None*		
clientId	int(10)		UNSIGNED	No	*None*		

Note: The **Attributes** field for the **clientId** in the *clients* table and the *reviews* table need to match (they must be the same). They should both be "UNSIGNED" OR both be blank. The same goes for the **invId** in the *clients* table and the *inventory* table.

Build the Relationships

1. Build the relationships between the *reviews*, *inventory*, and *clients* tables in the phpmotors database following the restraints shown:



Create the Folder and Controller

1. Create a "**reviews**" folder to store the controller for the application.

    - Create the controller in the "**reviews**" folder:
    - Use the typical name for the controller.
    - Add a comment to indicate this is the reviews controller
    - All controller components and processes must follow the typical patterns established in other controllers.
    - The needed processes are:
        1. Add a new review
        2. Deliver a view to edit a review.
        3. Handle the review update.
        4. Deliver a view to confirm deletion of a review.
        5. Handle the review deletion.
        6. A default that will deliver the "admin" view if the client is logged in or the php motors home view if not.

Create the Model

1. Create a "**reviews-model.php**" file for all database functionality. The model will need functions to:

    - Insert a review
    - Get reviews for a specific inventory item
    - Get reviews written by a specific client
    - Get a specific review
    - Update a specific review
    - Delete a specific review

2. Be sure it is required into the controller.

Add and Display Reviews

1. In the existing "vehicle detail" view provide text, that is clearly visible when the page loads, indicating that vehicle reviews can be seen at the bottom of the page.
2. Create a clear "Customer Reviews" heading after the vehicle content area, but before the footer.
3. Beneath the "Customer Reviews" heading show text indicating that a

review can be added by "logging in" and provide a link to the deliver the "login" view. If the client is already logged in, then provide the form for entering a review. The form must:

- Only provide space for the review to be written.
- Display the "screen name" (the first initial of the first name and the complete last name, with no spaces) in the form and not be editable. You'll need to use the PHP substr() function for this, refer to https://www.php.net/manual/en/function.substr.php)
- Include the inventory ID in a hidden field in the form.
- Include the client ID in a hidden field in the form.
- Include an "action" trigger name - value pair.
- Be directed to the reviews controller for processing.

4. If there are existing reviews for the vehicle, they should be queried from the database and displayed beneath the text or form described in step 3.
5. Reviews that are displayed in the vehicle detail view must show the most recent review listed first and the oldest review listed last.
6. Individual reviews must include 1) the review text, 2) the reviewer's screen name (the first letter of their firstname and the complete last name as a single string with no spaces - use the PHP substr() - https://www.php.net/manual/en/function.substr.php for this) and 3) the date of the review (you'll use the PHP date() - https://www.php.net/manual/en/function.date.php, and strtotime() - https://www.php.net/manual/en/function.strtotime). Each review must be visually distinguishable from other reviews.

Manage Reviews

Add new views as needed to accomplish the tasks listed below.

1. In the existing "Client Admin" view display a list of reviews (if any) that the logged-in client has written with the ability to update or delete the individual review.

    a. If the client opts to update a review, the review information must be displayed in the update form within a review update view for editing. Only the review text should be editable!
    b. If the review text is empty when the update form is submitted, the view should be returned, with the original review text restored and an error message displayed.
    c. When a review update is finished, the "Client Admin" view should be

delivered with an appropriate message indicating the outcome of the update.

d. If the client opts to delete a review, the review information should be displayed, but not be editable, for confirmation with a warning that the delete cannot be undone in a review delete view.
e. When the delete is completed, the "Client Admin" view should be delivered with an appropriate message and the list of remaining reviews should be displayed. The deleted review should no longer be part of the list.

Test, Test, Test

Just as with all activities and enhancements throughout the semester, you must test thoroughly to make sure things work, including:

1. A vehicle review can be added, but only by a logged-in client.
2. The newly added review is added to the "reviews" table of the phpmotors database along with the inventory ID and the client ID.
3. The vehicle review appears in the "vehicle detail" view for that specific vehicle.
4. The client's reviews appear in the "client admin" view after logging in.
5. That a review can be updated and deleted by the client.
6. That all views in the "reviews" application are valid HTML5 and CSS3 and responsive to differing screen sizes and easily read.

Submitting

When done testing and satisfied that everything is working as it should, do the following:

1. Work with two other class members to conduct a peer review of your finished project. They should use the provided grading matrix to review your project and you should review theirs.
2. Based on their feedback make whatever corrections seem appropriate to make your project better and to maximize its usability and your scores.
3. Submit your peer review files (the ones you did, not those done for you) to the **Final Project Peer Reviews** page.
4. Export the phpmotors database as an SQL file.
5. Save the SQL file into your "**phpmotors/sql"** folder.
6. Create a zip file of the entire **"cse340"** folder and name it "YOURNAME_FinalProject_CSE340.zip".

7. Create a video that demonstrates all the functionality that is listed in Objective 6 of the Grading Matrix. You will be showing the code and the website.
8. Save and publish the video to your YouTube channel as an **Unlisted** video and copy the URL link to the video.
9. Commit and push your changes to your private GitHub repository and copy the link to the repository.
10. Submit the zip file to the **Final Project Code Submission** page, and add the video link, and the GitHub link as comments.

Grading Matrix

Objective 1
Standard: Views are valid HTML5

- Randomly select 1 view from the final project. Validate for HTML5 compliance.

Standard: Views are valid CSS3

- Randomly select 1 view from the final project. Validate for CSS3 compliance.

Standard: Views and content are responsive to viewport size

- Randomly select 1 view from the final project. Resize the browser window. Content should adapt to display without the need for zooming or horizontal scrolling.

Standard: Views are usable and provide a consistent user experience

- As the project views are navigated ask yourself:
  1) Are the views both usable and consistent in appearance?
  2) Is the content easily read (large, clear fonts) and present in form inputs when updates or deletes are being carried out?

Objective 2
Standard: Control structures are implemented for adding, updating and deleting product reviews

- Examine the reviews controller, are control structures present *and operational* for adding, updating and deleting product reviews (including delivery of views to do the update or confirm the delete)?

Standard: Control structures are implemented for the "admin view" to display a list of reviews, if any, for management.

- Examine the code to determine if product reviews will be displayed in the "admin view". Is it present and does it work?

Objective 3

Standard: Models contain functionality for database interactions

- Check the "review" model. Are database functions present, and operational, to insert, select, update and delete product reviews in the database?

Standard: Controllers contain the logic layer and dictate the operational flow of the application

- Review the "review" controller for logical control:
    - 1) receiving input from views,
    - 2) as needed calling data via database interaction functions, building logical response messages and delivering views. Do all these operations occur solely in the controller and are they operational?

Standard: Views are reserved to interaction / presentation only

- Review the code for views used in the "review" application to insure that they only display content and do not contain controller logic or make direct calls to model-based functions.

Objective 4

Standard: The "reviews" database table meets the specifications

- Exam the "reviews" table of the **phpmotors** database. Does the table and fields meet the naming and ERD requirements?
- Is the "reviews" table being used for all review data?

Standard: SQL queries and PDO prepared statements are used for a complete CRUD implementation

- Check the functions in the "reviews" model. Are PDO prepared statments used and correct SQL statements present to INSERT, SELECT (these will use JOINS to get the correct data), UPDATE and DELETE data?

Objective 5

Standard: Client-side and server-side validation of all inputs is provided

- Attempt entering incorrect inputs to an update review form. Does there appear to be client-side validation in place?
- Use the no-validate plugin to bypass the client-side validation. Enter the data with intentional errors into the form and submit. Does there appear to be server-side validation?

Standard: Proper data types are checked and both sanitization and

validation are used

- Check controllers for use of input filtering/sanitizing/validation for all incoming data (this could be done in a variety of ways). Are these methods being used?

Standard: Error correction is pushed back to the user for correction

- When errors are entered in the application, are messages indicating the type of error displayed and the user forced to correct errors before the application will proceed?
- When errors are corrected does the application proceed and work correctly?

Objective 6

Standard: The project and overview video are complete and delivered on time

- The video URL, GitHub repo URL and zip of the entire “cse340” folder are submitted to the Final Project Code Submission page.
- The **phpmotors database** SQL file is included in the project "zip" file and importable without alteration.
- The video overview should not be longer than 10 minutes. The video shows the following points:
    - Adding a vehicle review using an appropriate form in the product detail view, but only when logged-in.
    - Show the newly added review in the "reviews" table of the phpmotors database.
    - Show the product review displayed in the "vehicle detail" view only for that vehicle.
    - Show the client's reviews in the "client admin" view after logging in.
    - Show that a review can be updated and deleted by the client.
    - Objective 1
        - Validate the HTML of one view in the "reviews" application.
        - Validate the CSS of a different view in the "reviews" application.
        - Show that one of the vehicle detail pages with a review is responsive.
        - Show that page with the client’s list of reviews is responsive.
    - Objective 2
        - Show the code for the reviews controller and point out the code that adds, updates and deletes a review.
        - Show the code that displays the product reviews in the admin page.

- Objective 3
  - Show the code in the reviews model file and point out the database functions to insert, select, update and delete product reviews in the database.
  - Show the code in the review controller that receives input from the view and calls a function from the model.
  - Show the code in the views used in the review functionality to show that no controller logic is used.
- Objective 4
  - Show the reviews table in phpMyAdmin. Show that the names of the columns match the naming convention used in the course.
  - Point out how all of the reviews data is stored in this table an no other table.
- Objective 5
  - Enter data into the review form and show that client-side and server-side validation work.
  - Show the code in the reviews controller that filters, sanitizes and validates the input from the form.
  - Show the error messages when data is input incorrectly, and then show the application working correctly when errors are fixed.

Standard: Communication within the application is error free

- As the php motors site is being used watch for spelling and grammar errors.