2703ICT - Assignment 2

Due Date: 9am Monday 11 October 2021 (Week 12)
Weight: 50%
Individual Assignment

Introduction
In assignment 2, we will create a ***review*** *application*. This application stores and display user reviews for a particular type of items (e.g., products/services/websites) of your choice. E.g., A review website for mobile phones. You are free to choose the type of items your website caters for.

You are required to use Laravel 8. Your implementation must use Laravel's migrations, seeders, models, ORM/Eloquent, route, controllers, validator, and view/blade. You are allowed to use all the features provided by Laravel.

Details
You have some freedom in designing the functionality of your website, however, it must satisfy the following requirements:

1. Users can **register** with this website. When registering, users must provide their name, email, and password. Furthermore, users must register as either a:
    - Member, or
    - Moderator.
2. Registered users can **login**. Login (or getting to the login page) can be done from any page. Once logged in, the username and the user type (member or moderator) will be displayed at the top of every page. A logged in user should be able to log out.
3. Users that have not logged in (**guests**) can see items and reviews, but cannot perform and credit, update, or delete.
4. **Members** can retrieve items and reviews, as well as post items and review. Member can also update and delete their own reviews.
5. **Moderators** can perform CRUD on all items and reviews.

Items	Guest	Member	Moderator
Create	No	Yes	Yes
Retrieve	Yes	Yes	Yes
Update	No	No	Yes
Delete	No	No	Yes

Table 1. Permission table for Items

Reviews	Guest	Member	Moderator
Create	No	Yes	Yes
Retrieve	Yes	Yes	Yes
Update	No	Own	Yes
Delete	No	Own	Yes

Table 2. Permission table for Reviews

6. An **item** must contain the item name, manufacture/business name, description for that item, a recommended retail price, and an URL to that item. All fields except URL must not be empty. Price must be greater than 0. The item name must be unique. An URL, if supplied, must be a valid URL (in terms of format).
7. A **review** must contain a creation date, a rating, and the review. A rating must be a number between 1 to 5, and review must have more than 5 characters.
    - A member/moderator can only post at most one review per item. After a new review is posted, the user will be redirected to the page containing the newly posted review.

8. **List**: There should be a page that lists all items, and allow user to click on an item to show the detail page for that item.
9. The **details page** should show all the information for that item, in addition it also displays all reviews for that item. When displaying review, all information for that review should be displayed. In addition, the name of the user who posted that review must also be displayed.
10. The **update** form for item and review should contain the old value.
11. When an item is **deleted**, all reviews for that item should also be deleted.
12. **Input validation** must be performed. If invalid input is detected, the appropriate error message must be displayed, along with the previous entered value.
13. **Access control**: actions that cannot be performed by user should not be displayed (e.g., if a user is not logged in, s/he should not see the form for posting a review, and a user should not see the edit button/link for the reviews they cannot edit etc). Appropriate access control checks should also be implemented on the server before performing an operation.
14. **Pagination** must be used to paginate reviews; where there are more than 5 reviews for an item (at least one item must have more than 5 reviews).
15. Users can upload **images** of an item. An item may have many images uploaded by different users. These images will be displayed in the details pages for that item. When displaying an image, the name of the user who uploaded the image should also be displayed. Only logged in user can upload item image. (Note: partial mark will be awarded if you allow only ONE photo per item.)
16. Users can vote to **like or dislike** reviews. Reviews that have more dislike than like would be displayed differently than the reviews with more likes than dislikes (you can decide how to display them differently). A user can only vote for a review once. As reviews are often retrieved, the implementation should be optimised for efficiency of retrieval of review (possibly at the expense of using more storage space).
17. A user can "***follow***" and "***unfollow***" reviewers/users. By following other reviewers, the currently logged in users can:
    - See a list of people the s/he has followed, and
    - See a list of reviews posted by the reviewer s/he has followed. For each review, the item which the review is for should also be indicated.
18. A **recommendation** feature which provides personalised recommendation for the current user, such as:
    - Items the current logged in user may like.
    - Reviewers the current logged in user may want to follow.

    You have the freedom to design and implement this feature. You will be judged on creativity/innovation, usefulness, and also technical competence.

Technical requirements

- Use Laravel's migration for database table creation.
- Use Laravel's seeder to insert default test data into the database. There should be enough initial data to thoroughly test the retrieval, update, and deletion functionalities you have implemented.
- Use Laravel's ORM/Eloquent to perform database operations. Only partial mark will be awarded for implementations using SQL or query builder.
- Proper security measures must be implemented.
- You must not implement any client side (html/Javascript) validation, so your server-side validation can be tested.
- Good coding practice is expected. This includes:
  - Naming: using consistent, readable, and descriptive names for files, functions, variables etc.
  - Readability: correct indenting/spacing of code.
  - Commenting: there should at least be a short description for each function.
  - View: proper use of template and template inheritance.

Documentation
Provide the following documentation in no more than 1 page:

1. An ER diagram for the database. Note: many-to-many relationships must be broken down into one-to-many.
2. Describe what you were able to complete, what you were not able to complete.
3. Reflect on the process you have applied to develop your solution (e.g. how did you get started, did you do any planning, how often do you test your code, how did you solve the problems you come across). What changes would you make for assignment 2 to improve your process?
4. If you have completed the recommendation feature, describe how you have implemented this feature.

This documentation should also be provided as a page in your website and linked to from the navigation menu.

Video
Create a video (with audio) to demonstrate the functionality of your application. In the video, you need to state your name, and then explain what your application does, and demonstrate each of the features. You do not need to discuss your code in this video (we will ask you to do this during in class demonstration). Your video should be around 3 minutes.

For further details of the requirements, refer to the marking rubric. **All requirements from both the assignment specification and marking rubric must be satisfied.**

Submission Requirements
You must submit the following items for the assignment:

1. A compressed file containing ALL source files in your submission (including all PHP code), but **excludes the vendor and node_modules directories**.
   - This file must be submitted via Learning@Griffith through the assignment 2 link.
     Note: Delete the *vendor* and *node_modules* directories before you compress the files. (Restore the vendor directory before your demonstration with the command: *composer update*. The node_modules directory can be restored by the command: *npm install*). Use the *zip* command to compress your assignment directory (see Lecture 1-3).
2. A PDF file containing your documentations.
3. A link/URL to your demonstration **video**. To do this, when submitting your assignment, click on 'Write Submission' and paste the link.
4. These files must be submitted via Learning@Griffith through the assignment 2 link.

Do not update your source code after you have submitted your work. During the demonstration, you need to show the last modified date of your file (on Elf, run the command: *ls -la* in your *controller and routes* directory).

Note: You are responsible for regularly backing up your work. Hence, if you lose your file due to not backing up, then expect to be heavily penalised.

Assignment Demonstration, and Marking
After you have completed your peer review, you must demonstrate and explain your work to your tutor in Week 12 lab to have your submission marked by your tutor and receive personalised feedback.

If you do not *demonstrate your assignment* to your tutor, your submission will be regarding as incomplete, hence you will not receive a mark for this assessment item!

Warning: We take student academic misconduct very seriously!

Appendix

The following user accounts must be created via Seeder for testing purpose. You are free to add other users for testing purpose.

Name	Moderator
Email	Moderator@a.org
Password	1234
Type	Moderator

Name	Chris
Email	Chris@a.org
Password	1234
Type:	Moderator

Name	Member
Email	Member@a.org
Password	1234
Type:	Member

Name	Cara
Email	Cara@a.org
Password	1234
Type:	Member

Name	Bob
Email	Bob@a.org
Password	1234
Type:	Member

Name	Fred
Email	Fred@a.org
Password	1234
Type:	Member