Assignment 1 – 2703ICT Web Application Development

Due Date: (Friday Week 7) 10th September 2021 5pm
Weight: 40%
Individual Assignment

The aim of this assignment is to build a simple web application with the basic building blocks (to a reasonable level). The only support from Laravel framework you can use are routing and templating (view). The understanding gained from this assignment provides a foundation for students to learn other frameworks or to develop new frameworks. In assignment 2, we will let you unleash the capabilities of Laravel.

Introduction
For this assignment you are required to build a fleet management application. This application stores information regarding vehicles, booking information and customers.

There is NO requirement in Assignment 1 to support user authentication.

Details
The name and design of the website can be to your choosing however it must satisfy the following requirements:

1. All pages must have a **navigation menu**, either across the top of the page or down the left or right column.
2. The **home page** must list all vehicles in the database by their Rego[1]. Clicking on a vehicle will bring up the details page of that vehicle.
3. The **details page** must display ALL the information for that vehicle, plus ALL the bookings for that vehicle.
4. A **vehicle** must have a rego, model, year and odometer value. A rego must be an alphanumeric string of 6 characters (sorry, we don't cater for personalised number plate). Odometer value must be a number. In addition, you need to add one more sensible attribute of your choice to vehicle.
5. **Clients** are people that can book vehicles. The following information about a client must be store: name, age, drive's license number, and license type. In addition, you need to add one more sensible attribute of your choice to client. There is a page that display all clients and their details.
6. **Create**: Users can create a new vehicle or client (you pick one to implement).
7. **Delete**: Users can delete an existing vehicle or client (you pick one). The booking associated with that vehicle or client must also be deleted.
8. **Update**: Users can update either an existing vehicle or client (you pick one). Existing values must be displayed in the update form.
9. **Validation**: In either Create or Update, you must implement validations which check for:
    a. An input that is of type string must have at least a certain number of characters (e.g. a rego must be at least 6 characters), and
    b. An input that is a number that is within a range (e.g. age must be greater than 17 and less than 99).

[1] Rego is short for vehicle registration number that can uniquely identify a vehicle. In Queensland, the rego is usually a string of 6 characters.
https://en.wikipedia.org/wiki/Vehicle_registration_plates_of_Queensland

10. There is a page for clients to **book a vehicle**. In this page, there is a drop-down list of all the clients (name), a drop-down list of all the vehicles (rego) to select from. In addition, the user of this page needs to enter a starting date and time, and a returning date and time. Once this information is submitted, the booking information will be stored (in the database). Note: if you are not taking on requirements 14, 15 and 16, then date and time can just be stored as a string/varchar.
11. The booking information to be displayed in the **details page** for a vehicle (see requirement 3) must contain the client's name, license number, and the starting and returning date and time.

The requirements below are more difficult, and we would recommend you attempt the requirements below only after you have implemented the requirements above.

12. The website needs to handle **vehicle return**. When a client returns a vehicle, the distance driven needs to be entered and update to the vehicle's odometer value. As the associated booking is now fulfilled, this booking should not be displayed in the details page.
13. There is a page that lists all vehicles in the order of **how often** they are booked, i.e. the car with the most number of bookings is displayed first.
14. There is a page that list all vehicles in the order of the **total amount of time** they are booked (i.e. the car that is booked for the longest time is displayed first). The total amount of time booked for each vehicle must also be displayed.
15. **Booking validation 1**: When booking a vehicle, the starting date and time must be a date and time in the future. The returning date and time must be later than the starting date and time. This validation must be checked on the server side, and error must be appropriately handled.
16. **Booking validation 2**: A new vehicle booking must not overlap in date and time with an existing booking. If a submitted booking overlaps with an existing booking, then the booking will not be successful, and the user will be informed.

Technical requirements

1. This assignment must be implemented using Laravel. However, database access should be implemented via raw SQL and executed through Laravel's DB class. You are not to use Laravel's ORM (ORM will be used for assignment 2). You must implement your own input validation (instead of using Validator).
2. An SQL file should be used to create tables and insert initial data. There should be enough initial data to thoroughly test the retrieval, update, and deletion functionalities you have implemented.
3. You must implement your own validation (you should not use Laravel's validation feature). Validation errors message must be displayed within the view. Note: validation must be done on the server side (you should know why). To demonstrate this, you should not have any client-side validation.
4. Proper security measures must be implemented, e.g. perform HTML and SQL sanitisation etc. You should be able to explain the security measures you have implemented.
5. Template inheritance must be properly used.
6. 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.

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 booking validation 2 (requirement 16) describe what conditions did you check to ensure bookings do not overlap.

This documentation should be provided as a page in the 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 state your name and student number, and then <u>explain</u> 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

Your submission should consist of:
1. A compressed file (.zip) containing your application (including all source code). See Week 1 lecture regarding how to zip a directory and download file.
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.

These files must be submitted via Learning@Griffith through the assignment 1 link.

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.

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 *routes* directory).

Assignment Demonstration and Marking

You must demonstrate and explain your work to your tutor in Week 8 lab to have your submission marked by your tutor and receive personalised feedback.

<u>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!</u>

Warning: We take student academic misconduct very seriously!