Submission Worksheet

Submission Data

Course: IT202-007-F2025
Assignment: IT202 Milestone 2
Student: Dylan G. (dg599)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 802.00/802.00 (100.00%)
Received Grade: 0.00/802.00 (0.00%)
Started: 11/24/2025 6:42:19 PM
Updated: 11/24/2025 11:14:33 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT202-007-F2025/it202-milestone-2/grading/dg599
View Link: https://learn.ethereallab.app/assignment/v3/IT202-007-F2025/it202-milestone-2/view/dg599

Instructions

- Overview Link: https://youtu.be/t6X8dzZt_a0

1. Refer to Milestone2 of this doc: https://docs.google.com/document/d/1XE96a8DQ52Vp49XACBDTNCq0xYDt3kF29cO88EWVwfo/view
2. Ensure you read all instructions and objectives before starting.
3. Ensure you've gone through each lesson related to this Milestone
4. Switch to the Milestone2 branch
    1. `git checkout Milestone2` (ensure proper starting branch)
    2. `git pull origin Milestone2` (ensure history is up to date)
5. Fill out the below worksheet
    - Ensure there's a comment with your UCID, date, and brief summary of the snippet in each screenshot
    - Ensure proper styling is applied to each page
    - Ensure there are no visible technical errors; only user-friendly messages are allowed
6. Once finished, click "Submit and Export"
7. Locally add the generated PDF to a folder of your choosing inside your repository folder and move it to Github
    1. `git add .`
    2. `git commit -m "adding PDF"`
    3. `git push origin Milestone2`
    4. On Github merge the pull request from `Milestone2` to `dev`
    5. On Github create a pull request from `dev` to `prod` and immediately merge. (This will trigger the `prod` deploy to make the heroku prod links work)
8. Upload the same PDF to Canvas
9. Sync Local
10. `git checkout dev`
11. `git pull origin dev`

Section #1: (2 pts.) Data

Progress: 100%

☰ Task #1 (0.50 pts) - API Data Table

Task #1 (0.00 pts.) - API Data Table

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the table(s) structure of the table made to hold your API
- There should be at least 3 properties/fields beyond `id`, `created`, `modified`, and respective `is_api` or `api_id` columns
- Logical types and constraints should be set
- Show at least a few records from each API table (samples should include both API fetched data and custom entries noted by an `api_id` or `is_api column)

id int	api_id varchar(100)	is_api tinyint(1)	make varchar(100)	model varchar(100)	year int	type varchar(50)	image_url text	description text	user int	created timestamp	modified timestamp
1	(NULL)	0	Ford	Mustang	2020	Sedan	https://www.ford.com/cars/		5	2025-11-24 15:49:39	2025-11-24 15:14:18
3	subaru-wrx-sti-2.4-271hp	1	Subaru	WRX (VB) II	2025	Sedan	https://www.auto-data.net/	2.4 (271 Hp) AWD SPT | Sed	5	2025-11-24 19:53:42	2025-11-24 19:53:42
4	subaru-wrx-sti-2.4-271hp	1	Subaru	WRX (VB) II	2025	Sedan	https://www.auto-data.net/	2.4 (271 Hp) AWD | Sedan, i	5	2025-11-24 19:53:42	2025-11-24 19:53:42
5	subaru-wrx-sedan-va-fastlift	1	Subaru	WRX Sedan (VA) (facelift 20	2025	Sedan	https://www.auto-data.net/	2.0 (268 Hp) AWD CVT | Sed	5	2025-11-24 19:53:42	2025-11-24 19:53:42
6	subaru-wrx-sedan-va-fastlift	1	Subaru	WRX Sedan (VA) (facelift 20	2025	Sedan	https://www.auto-data.net/	2.0 (268 Hp) AWD | Sedan, i	5	2025-11-24 19:53:42	2025-11-24 19:53:42
7	subaru-wrx-sti-facelift-2018	1	Subaru	WRX STI (facelift 2018)	2025	Sedan	https://www.auto-data.net/	5299 2.5 (341 Hp) AWD | Se	5	2025-11-24 19:53:43	2025-11-24 19:53:43
8	subaru-wrx-sti-facelift-2018	1	Subaru	WRX STI (facelift 2018)	2025	Sedan	https://www.auto-data.net/	F210 Final Edition 2.0 (309 H	5	2025-11-24 19:53:43	2025-11-24 19:53:43
9	subaru-wrx-sti-facelift-2018	1	Subaru	WRX STI (facelift 2018)	2025	Sedan	https://www.auto-data.net/	2.5 (310 Hp) AWD | Sedan, i	5	2025-11-24 19:53:43	2025-11-24 19:53:43
10	subaru-wrx-sti-facelift-2018	1	Subaru	WRX STI (facelift 2018)	2025	Sedan	https://www.auto-data.net/	2.5 (300 Hp) AWD | Sedan, i	5	2025-11-24 19:53:43	2025-11-24 19:53:43
11	subaru-wrx-sedan-va-2.0-2i	1	Subaru	WRX Sedan (VA)	2025	Sedan	https://www.auto-data.net/	2.0 (268 Hp) AWD | Sedan, i	5	2025-11-24 19:53:43	2025-11-24 19:53:43
12	subaru-wrx-sti-2.5-300hp-a	1	Subaru	WRX STI	2025	Sedan	https://www.auto-data.net/	2.5 (300 Hp) AWD | Sedan, i	5	2025-11-24 19:53:43	2025-11-24 19:53:43

database table

```
-- Dump completed on 2025-11-24 19:53:43
```

sql file

Saved: 11/24/2025 7:52:02 PM

Part 2:

Progress: 100%

Details:

- Add a direct link to your sql file from Github (should end in `.sql`)

URL #1

https://github.com/dylangarca/dg599-IT2025/0/4

URL

https://github.com/dylangarca/dg

Saved: 11/24/2025 7:52:02 PM

Part 3:

Progress: 100%

Details:

- Briefly note what each field/property represents

Your Response:

The id field is the primary key and is a unique identifier for each car in the database. API id is a unique id that is provided by the car api and this field is null when the car is created manually. is api field is showign whether the car is added through the api witha 1 or if the car was created manually with a 0. make is car's brand, model is the specific model of the car, year is the production of the year, type is category of the car, image url is where the url link of the image of the car, description is a short description of the car. user id is the id of the user who added the car into the database. created and modified are timestamps of when the car was created or modified. idx make,year, and api id are indexes to make the queiries quicker.

Saved: 11/24/2025 7:52:02 PM

Task #2 (0.50 pts.) - Data Transformation

Progress: 100%

Part 1:

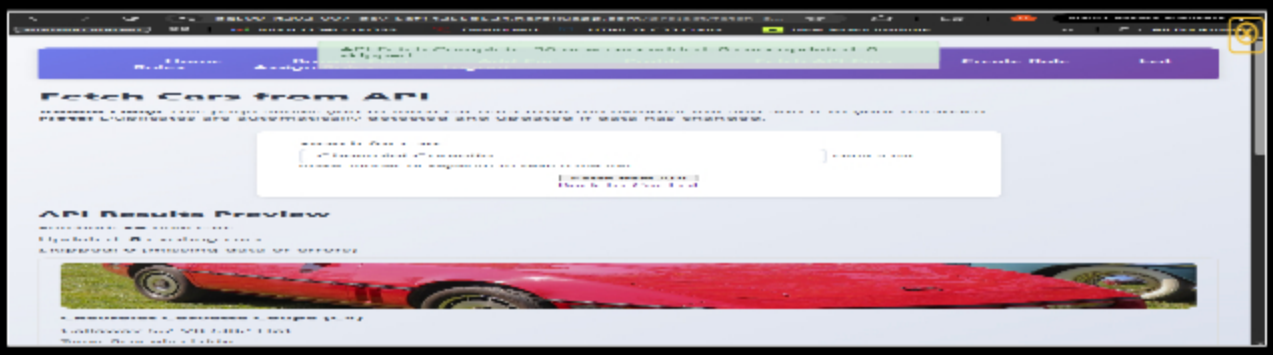
Progress: 100%

Details:

- Show the code that handles the data transformation/conversion (from the API)
- Show a Heroku dev example of API getting fetched and loaded into the DB

process api code p1

process api code p2



dev example of fetch

Saved: 11/24/2025 8:02:15 PM

Part 2:

Progress: 100%

Details:

- Include Heroku prod link to API fetch page
- Include pull request link for this feature

URL #1
https://dg599-it202-007-dev-5af14255b524.herokuapp.com/project/fetch_cars.php

URL
https://dg599-it202-007-dev-5af1

URL #2
https://github.com/dylangarca/dg599-IT202-007/pull/1

URL
https://github.com/dylangarca/dg

Saved: 11/24/2025 8:02:15 PM

Part 3:

Progress: 100%

Details:

- Will you be using all the data or just a subset?
- Do you need to convert any data or extract pieces of a property's value?
- Briefly explain the transformation/extraction logic to get the API data to the shape your application intends to use

Your Response:

I am using a subset of the data, id is mapping to api id, the title gets split into make and model, content is used for the description of the car, additional is used to detereming the car type, image is for the image url. I extract the make and model of the cars from the title string and then i convert the additional field has phrases and i take certain parts of that feild to get the a clean type of the car. Then i combine the content and addiotnal field to make the description. So the goal is get the messy api data and turn it into clean values like when we split the title into the make an model field, detect the car type from additional by using keyword mathcing, and also sanitizing inputs.

Saved: 11/24/2025 8:02:15 PM

Task #3 (0.50 pts.) - API Duplicates

Progress: 100%

Details:

- Consider how duplicate entries are handled
  - What happens if you get the same result from the API for an entity that exists but has been unmodified?
  - What happens if you get the same result from the API for an entity that exists but has been modified?

Your Response:

So based off the two given cases, in the first case of the entity exist but the data wasn't modified what will happen is that Update will execute but all the values are going to stay the same because there is no change, all this case will do is increase the updated count. The user will see that no new cars were added, and shows X cars updated and Y cars skipped based on the number of cars. Now in the second case of the entity exist but the api data has been modified it will execute an update and update the fields that got changed and it will display a similar string to the user like the previous case.

Saved: 11/24/2025 8:07:44 PM

Task #4 (0.50 pts.) - API Fetching

Progress: 100%

Details:

- Consider how your application will trigger this data fetch
  - (i.e., periodic via cron job, poor man's cron job, on app start/awake, etc)
  - (i.e., User or Admin triggered via a specific page/action)
  - (i.e., Lazy loaded during searches for stale/missing content)
  - What happens if you get the same result from the API for an entity that

exists but has been modified?

Your Response:

The data fetch is occuring only when an admin vists the admin and fetches a car. So it is Admin triggered and is an on demand fetch. If the cars that were fetched already exists but the data was modified what will happen is that it will call the update and change the data to the newest data.

Saved: 11/24/2025 8:10:21 PM

Section #2: (150 pts.) Data Creation

Progress: 100%

Task #1 (60 pts.) - Validations

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the html validations (code)
- Show the javascript validations (code)
- Show the php validations (code)

html validation

```
<!--javascript
function validate(form) {
    let isValid = true;

    // Get form values
    const make = form.make.value.trim();
    const model = form.model.value.trim();
    const year = parseInt(form.year.value);
    const currentYear = new Date().getFullYear();

    // Validate make
    if (!make || make.length < 2) {
        alert('Make must be at least 2 characters long');
        isValid = false;
    }

    // Validate model
    if (!model || model.length < 1) {
        alert('Model is required');
        isValid = false;
    }

    // Validate year
    if (!year || year < 1900 || year > currentYear + 1) {
        alert('Please enter a valid year between 1900 and ' + (currentYear + 1));
        isValid = false;
    }

    return isValid;
}
//-->
```

js validation

```
// validation
if (empty($make)) {
    flash('Make is required.', 'danger');
    $hasError = true;
}

if (empty($model)) {
    flash('Model is required.', 'danger');
    $hasError = true;
}

if (empty($year) || $year < 1900 || $year > date('Y') + 1) {
    flash('Please enter a valid year between 1900 and ' . (date('Y') + 1), 'danger');
    $hasError = true;
}

if (!$hasError) {
    $db = getDB();
    $stmt = $db->prepare("INSERT INTO Cars (make, model, year, type, image_url, description, user_id, is_act
                          VALUES (:make, :model, :year, :type, :image_url, :description, :user_id, 1)");

    try {
        $stmt->execute([
            ":make" => $make,
            ":model" => $model,
            ":year" => $year,
            ":type" => $type,
            ":image_url" => $image_url,
            ":description" => $description,
            ":user_id" => $user_id
        ]);
        flash("Car added successfully!", "success");
        // redirect to cars page after successful insert
        header("Location: /Project/cars.php");
        exit;
    } catch (Exception $e) {
        flash("Error creating car: " . $e->getMessage(), "danger");
        error_log("Error creating car: " . var_export($e, true));
    }
}
?>
```

php validation

Saved: 11/24/2025 8:27:12 PM

Part 2:

Progress: 100%

Details:

- Briefly explain the html validations used
- Briefly explain the javascript validation logic
- Briefly explain the php validation logic

Your Response:

For the html form the key validation attributes that were used are required and i used this attribute for make,model, and year, this make sure that the user can't submit the form without these important fields. The next attribute is maxlength these are used to limit the amount of characters in the make, model, and type field. The next attribute is type number and this is used so that the user can not input characters into the year field. another type that is used is type url and i use this to make sure that the format for the url is correct. Then the last attribute for html is min 1900 and max current date + 1 and this is ensuring that the user enters a valid year range. For the js validation we have validate(form) and this is ensuring that before submitting the form the valjues of make, model, and year are valid inputs. So for make it checks if there are at least 2 characters, for model it is checking that is not empty, and then also making sure that the year is in the correct range. And if any of those three fail it will flash a warning message. For the php form the script is getting the values from the sanitize function, it is checking that make, model is not empty and year is not empty and it is the correct range. if any of the validations fail it will give a flash message, if the validation passes then it will create the car and add in the cars table.

Saved: 11/24/2025 8:27:12 PM

Task #2 (60 pts.) - Examples

Progress: 100%

--- Task Collapsed ---

Task #3 (30 pts.) - Links

Progress: 100%

Details:

- Include the heroku prod link to this page
- Include pull request link for this feature

URL #1
https://dg599-it202-007-prod-0134f160fb38.herokuapp.com/project/create_car.php

URL
https://dg599-it202-007-prod

URL #2
https://github.com/dylangarca/dg599-IT202-007/pull/41

URL
https://github.com/dylangarc

Saved: 11/24/2025 7:14:42 PM

Section #3: (150 pts.) Data List Page (Many Items)

Progress: 100%

Task #1 (60 pts.) - Code

Progress: 100%

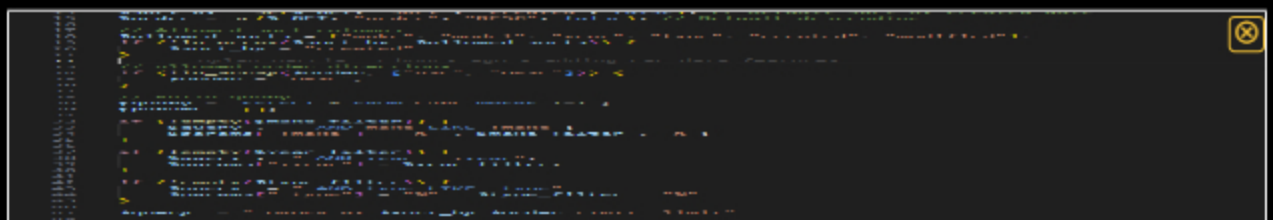
Part 1:

Progress: 100%

Details:

- Show the code that handles the filter/sort logic
- Show the code that generates the list/grid output
- Styling must be applied

filter/sort logic p1



filter/sort logic p2

output code

Saved: 11/24/2025 8:47:06 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the filter/sort logic works and formulates the results
- Briefly explain how the list/grid output is generated
- Briefly note the styling choices

Your Response:

For the logic the code is buidling a dynamic SQL queury based on the filters that get filled out by the user, then the code will validate all the inputs, it also will bind the parameters for the query securely, and lastly the code fetches the filtered/sorted list of cars. For the output, after the SQL query is run it is checking the cars array to see if it is empty and if so it will display a message of no results available but if it contains data then it will loop through the array then for each car a card is created and it shows an image of a car, the make, model, year, type, and wheter if the car was created by the api or manually. For the styling i use mainly bootstrap with light custom styling for a clean card look, spacing a readability.

Saved: 11/24/2025 8:47:06 PM

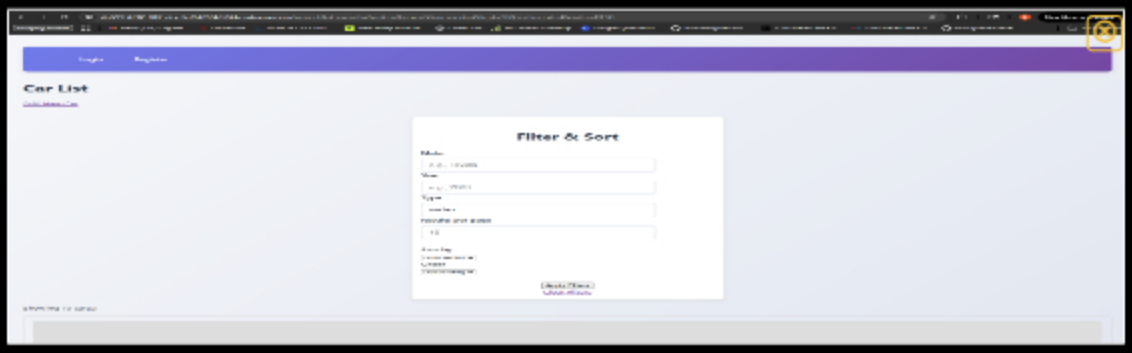
Task #2 (60 pts.) - Examples

Progress: 100%

Details:

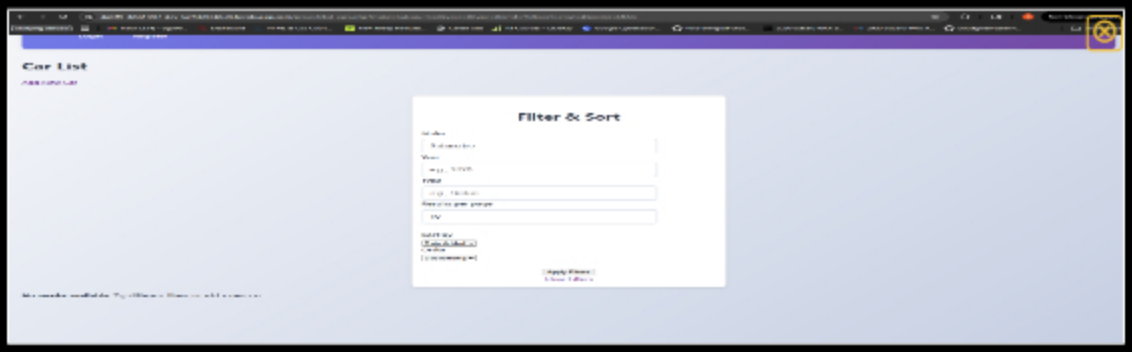
- Show a few variations of filter/sort applied (including no records)
- Each list item should be a summary

- Each list item should have the following links
  - A link to a single view of the specific entity (i.e., a details page)
  - A link to delete this entity (this may be an admin-only functionality, but it should be present for the respective role)
  - A link to edit this entity (this may be an admin-only functionality, but it should be present for the respective role)
- Ensure heroku url is visible



sedan sort p1

sedan sort p2



no results

Saved: 11/24/2025 8:53:52 PM

Task #3 (30 pts.) - Links

Progress: 100%

Details:

- Include the heroku prod link to this page
- Include pull request link for this feature

URL #1
https://dg599-it202-007-dev-5af14255b524.herokuapp.com/project/list_cars.php

URL
https://dg599-it202-007-dev-!

URL #2
https://github.com/dylangarca/dg599-IT202-007/pull/41

URL
https://github.com/dylangarc

Saved: 11/24/2025 8:54:44 PM

Section #4: (150 pts.) View Details Page (Single Item)

Progress: 100%

Task #1 (60 pts.) - Code

Progress: 100%

Part 1:

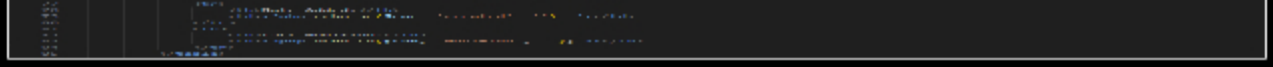
Progress: 100%

Details:

- Show the code that handles loading of the record (and how the id retrieved for usage)
  - Include the code that handles invalid/missing ids
- Show the code that generates the entity output
- Styling must be applied

```
<?php
// dgsuy 11/24
require(__DIR__ . "/../../partials/nav.php");
$car_id = (int)se($_GET, "id", 0, false);

if ($car_id <= 0) {
    flash("Invalid car ID", "danger");
    die(header("Location: list_cars.php"));
}

$db = getDB();
$stmt = $db->prepare("SELECT * FROM Cars WHERE id = :id LIMIT 1");

try {
    $stmt->execute([":id" => $car_id]);
    $car = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$car) {
        flash("Car not found", "danger");
        die(header("Location: list_cars.php"));
    }
} catch (Exception $e) {
    flash("Error fetching car details", "danger");
    error_log("Error fetching car: " . var_export($e, true));
    die(header("Location: list_cars.php"));
}
?>
```

code that handles loading of the record



code that generates entity

code that generates entity

Saved: 11/24/2025 9:21:43 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the loading logic works
- Briefly explain how the entity output is generated
- Briefly note the styling choices

Your Response:

The loading logic works by, the view cars page starts by getting the id from the url for example in the corvette example the id of that specific car was 94 the code then prepared a SQL queury to get the exact record mathcing that spefic id. Now if in either the first part the id is missing or not valid from the url it will provide an error message and redirect the user back to the list cars page, if the car cannot be found in the second part where it searchs for the record it will display an error message and redirect the user back to the list cars page. This process make sure that only valid and existing data records of cars are shown. For the entity output, once the car data is retrieved the html section will display the card with the data. So the output is using the se() function to securely echo the data values from the database to the page. The page will first show the image and then under it will display the car's year,make, model, description, created, and modified datestamps. The page also has logic wheter the car was fetched with the api or if it was manually added. THen there are also links at the bottom for edit,delete, and back. The styling choices are from bootstrap styling and inline css to give the page a clean look and modern as well and it also ensures that the content is readable and organized.

Saved: 11/24/2025 9:21:43 PM

Task #2 (60 pts.) - Examples

Progress: 100%

Details:

- Show a few examples of different entities
- Each entity item should have more details than the summary view (if possible)
- Each entity should have the following links
  - A link to edit the single entity (this may be an admin-only thing, but it should be present for the respective role)
  - A link to delete the single entity (this may be an admin-only thing, but it should be present for the respective role)

detailed view

detailed view p2

Saved: 11/24/2025 9:00:59 PM

Task #3 (30 pts.) - Links

Progress: 100%

Details:

- Include the heroku prod link to this page
- Include pull request link for this feature

URL #1
https://dg599-it202-007-dev-5af14255b524.herokuapp.com/project/view_car.php?id=94

URL
https://dg599-it202-007-dev-

URL #2
https://dg599-it202-007-dev-5af14255b524.herokuapp.com/project/view_car.php?id=10

URL
https://dg599-it202-007-dev-

URL #3
https://github.com/dylangarca/dg599-IT202-007/pull/41

URL
https://github.com/dylangarc

Saved: 11/24/2025 9:01:15 PM

Section #5: (150 pts.) Edit Page

Progress: 100%

Task #1 (60 pts.) - Code

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the code that handles loading of the record (and how the id retrieved for usage)
  - Include the code that handles invalid/missing ids
- Show the code that generates the form with the correct data types and populates it
- Show the html, javascript, and php validation (should be similar to create)
- Styling must be applied

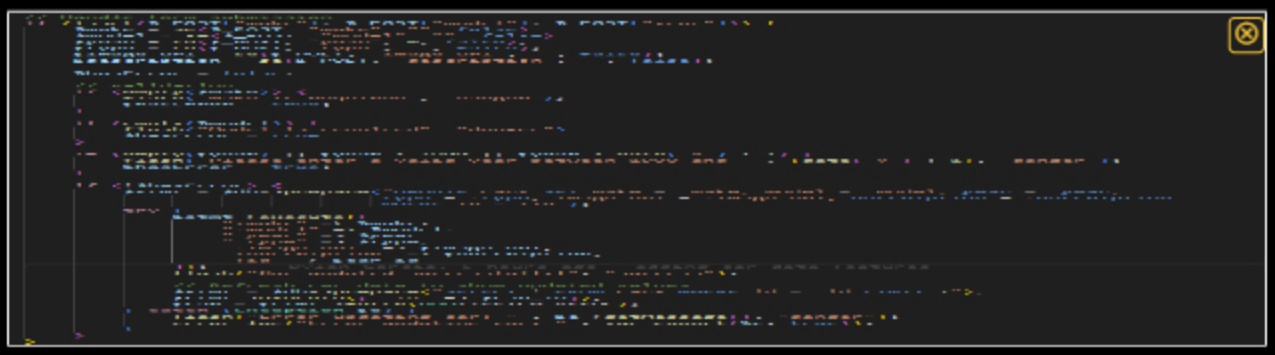
```php
$car_id = (int)se($_GET, "id", 0, false);

if ($car_id <= 0) {
    flash("Invalid car ID", "danger");
    die(header("Location: list_cars.php"));
}

$db = getDB();

// fetch existing car data
$stmt = $db->prepare("SELECT * FROM Cars WHERE id = :id LIMIT 1");
try {
    $stmt->execute([":id" => $car_id]);
    $car = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$car) {
        flash("Car not found", "danger");
        die(header("Location: list_cars.php"));
    }
} catch (Exception $e) {
    flash("Error fetching car details", "danger");
    error_log("Error fetching car: " . var_export($e, true));
    die(header("Location: list_cars.php"));
}
```

code that handles loading of the record and the invalid/missing ids

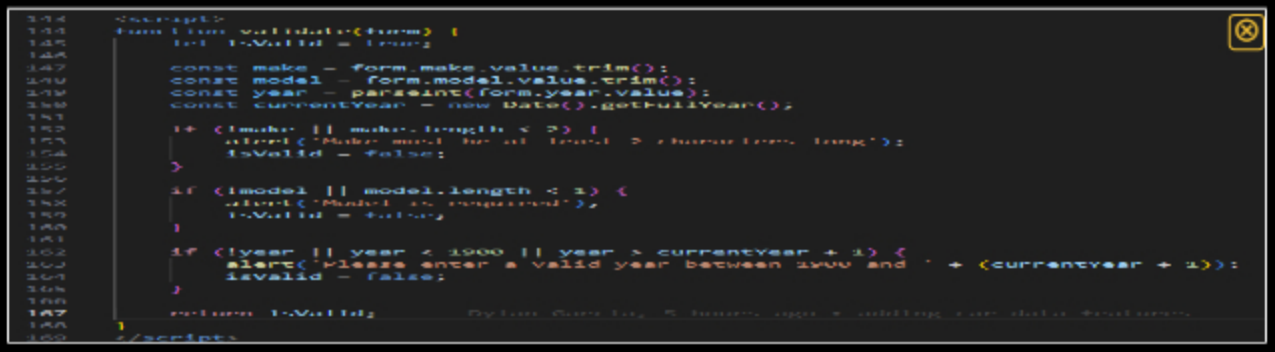
```
[illegible]
```

code that generates page

html validation



php validation



```
<script>
function validate(form) {
    let isValid = true;

    const make = form.make.value.trim();
    const model = form.model.value.trim();
    const year = parseInt(form.year.value);
    const currentYear = new Date().getFullYear();

    if (!make || make.length < 2) {
        alert('Make must be at least 2 characters long');
        isValid = false;
    }

    if (!model || model.length < 1) {
        alert('Model is required');
        isValid = false;
    }

    if (!year || year < 1900 || year > currentYear + 1) {
        alert('Please enter a valid year between 1900 and ' + (currentYear + 1));
        isValid = false;
    }

    return isValid;
}
</script>
```

js validation

Saved: 11/24/2025 10:17:48 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the loading logic works
- Briefly explain how the form is generated and submission is handled
- Briefly note the styling choices
- Note what fields are editable and why the choice was made for them (it's likely not all fields are editable, especially `id`, `created`, `modified`, etc)

Your Response:

For the loading logic it is first getting the if from the page url and then based off of that id it is validating whether it is a valid id and if not it will give an error message and redirect the user to

the lists page. After it make sure that the id is valid it will then query the database for the specific id and get the car data again if no record is there for the car then it will give the user an error message and redirect them to the list page. For the form generation it is populated with the existing car data and the form will include the proper input types for each field and the proper contstraints like required and maxlength. For the submission handling after the succeful update the car data will be reloaded and it will reflect the changes. For the styling choices i use bootsrap classes and minimal css styles to keep the same modern and readable appearance. the editable fields are only the fields that the user is able to access when adding their car so that is the make, model, year, type, image_url, and description. The user will not have access to change the id, created, or modified fields.

Saved: 11/24/2025 10:17:48 PM

Task #2 (60 pts.) - Examples

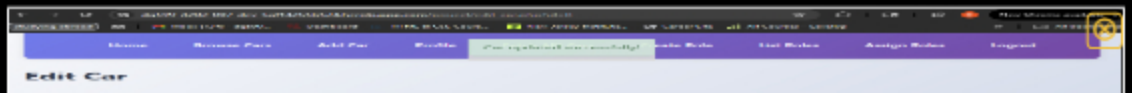
Progress: 100%

Details:

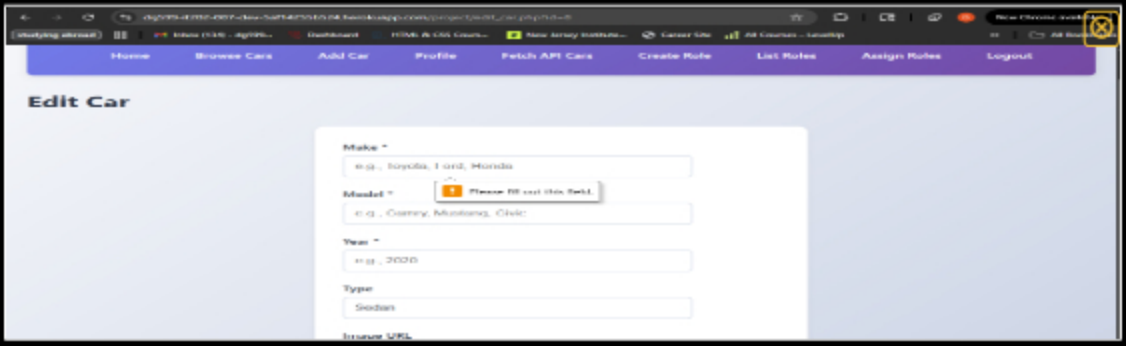
- Show a before and after of updating an entity (clearly caption)
- Successful update should have an appropriate message shown
- The updated data should be shown in the form
- Any errors should have user-friendly messages shown
- Show some examples of validation errors (html, javascript, php)

before

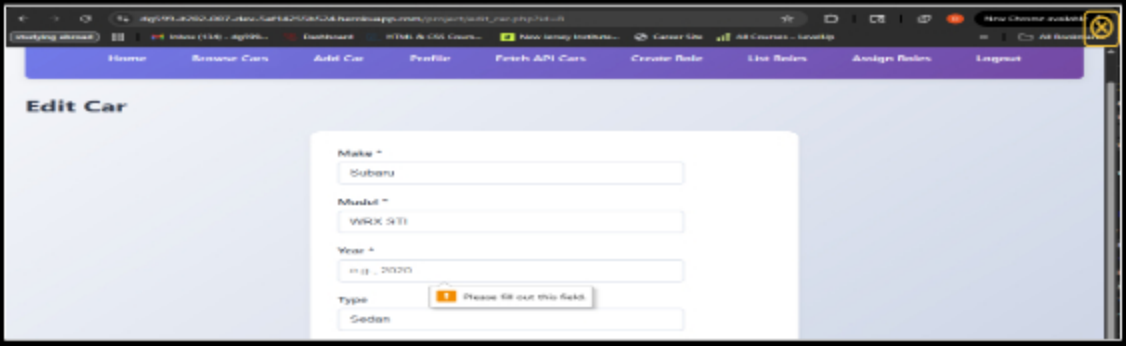
after



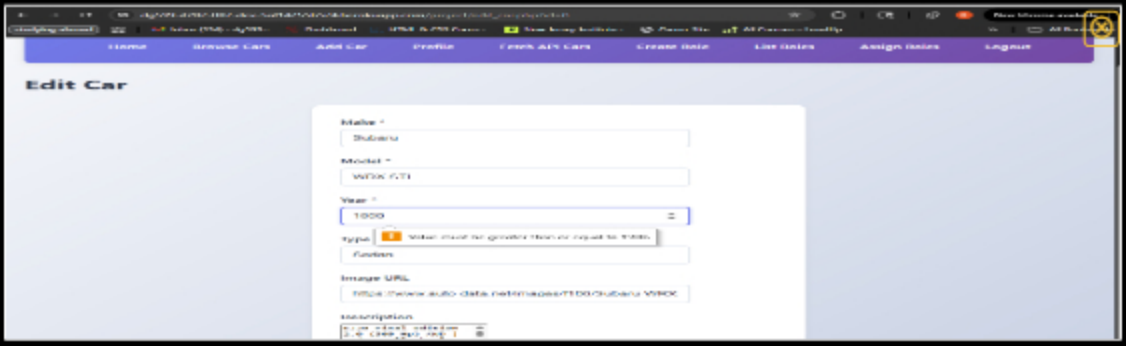
successful update



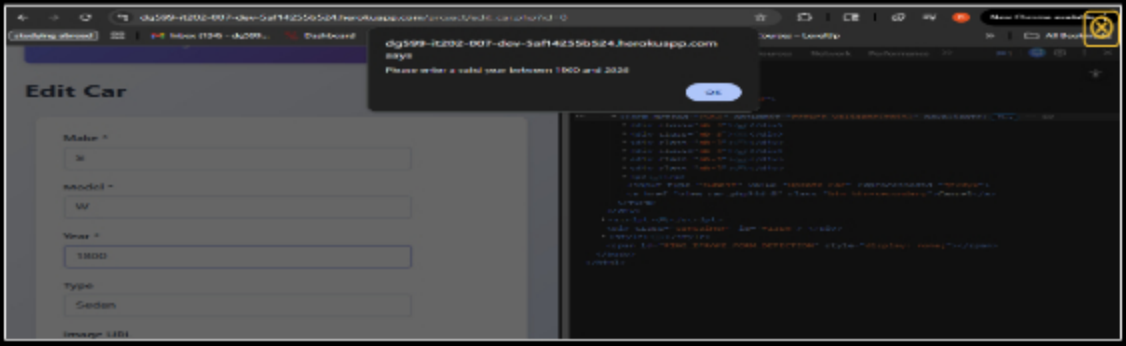
html validation p1



html validation p2

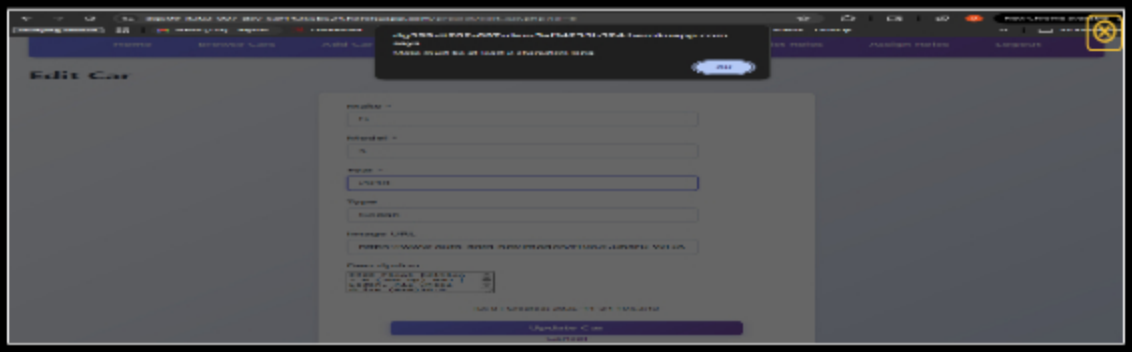


html validation p3

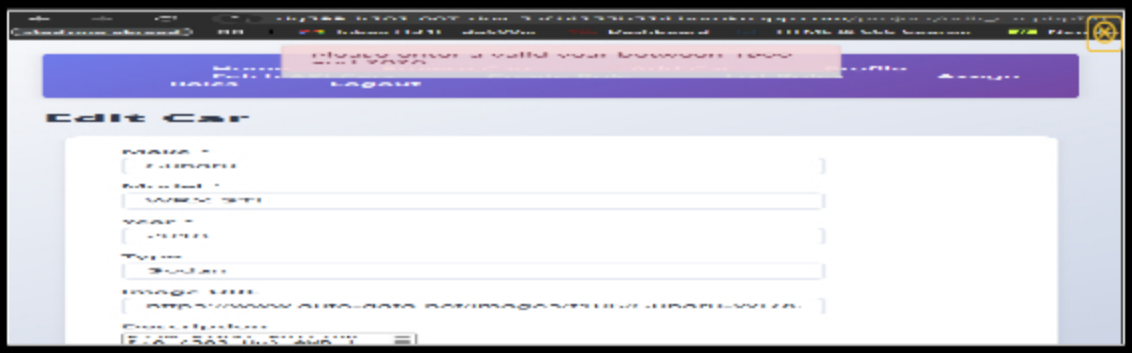


js validation

js validation



js validation



php validation

Saved: 11/24/2025 10:32:50 PM

Task #3 (30 pts.) - Links

Progress: 100%

Details:

- Include the heroku prod link to this page
- Include pull request link for this feature

URL #1
https://dg599-it202-007-dev-5af14255b524.herokuapp.cproject/view_car.php?id=8

URL
https://dg599-it202-007-dev-!

URL #2
https://github.com/dylangarca/dg599-IT202-007/pull/41

URL
https://github.com/dylangarc

Saved: 11/24/2025 10:33:21 PM

Section #6: (150 pts.) Delete Logic

Progress: 100%

Task #1 (60 pts.) - Code

Progress: 100%

Part 1:

Progress: 100%

Details:

- Show the code that handles deletion of the record (and how the id retrieved for usage)
  - Include the code that handles invalid/missing ids
- Handle any necessary roles/permissions (i.e., it's not likely that anyone can delete any entity they choose)
- Show the redirect logic that goes back to the previous page (where the route came from), if it was a list page, it should maintain the filter/sort criteria

```php
    $car_id = (int)($_GET["id"] ?? 0);

    if ($car_id <= 0) {
        flash("Invalid car ID", "danger");
        die(header("Location: list_cars.php"));
    }

    $db = getDB();

    // Fetch the car to check permissions
    $stmt = $db->prepare("SELECT * FROM Cars WHERE id = :id LIMIT 1");
    try {
        $stmt->execute([":id" => $car_id]);
        $car = $stmt->fetch(PDO::FETCH_ASSOC);

        if (!$car) {
            flash("Car not found", "danger");
            die(header("Location: list_cars.php"));
        }
    } catch (Exception $e) {
        flash("Error fetching car details", "danger");
        error_log("Error fetching car for delete: " . var_export($e, true));
        die(header("Location: list_cars.php"));
    }
```

delete code that handles deletion

```php
// Permission check: User can only delete their own cars, OR admin can delete any
$can_delete = false;
if (has_role("Admin")) {
    $can_delete = true; // Admin can delete anything
} elseif ($car["user_id"] == get_user_id()) {
    $can_delete = true; // User can delete their own cars
}

if (!$can_delete) {
    flash("You don't have permission to delete this car", "danger");
    die(header("Location: list_cars.php"));
}

```

delete code that handles necessary roles/permission

```php
// Redirect back to list page, preserving filters if they exist
$redirect_url = "list_cars.php";
if (!empty($_SERVER['HTTP_REFERER']) && strpos($_SERVER['HTTP_REFERER'], 'list_cars.php') !== false) {
    // Extract query string from referrer to preserve filters
    $referer_parts = parse_url($_SERVER['HTTP_REFERER']);
    if (isset($referer_parts['query'])) {
        $redirect_url .= "?" . $referer_parts['query'];
    }
}

die(header("Location: $redirect_url"));
?>
```

delete code redirect logic

Saved: 11/24/2025 10:45:43 PM

Part 2:

Progress: 100%

Details:

- Briefly explain how the deletion logic works
- Is it a soft or hard delete?
- Note any permissions/rules applied.
- Briefly explain how the redirect maintains the previous route and any filter/sort data

Your Response:

For the deletion logic it is getting the car id from the url and then gets the information from the database off of the car id it gets. It will also make sure that the car exists and is a valid car id. Then if the car exists and the user has the correct permission/role the delete command will execute and remove the data of that car from the database. I used a hard delete. It will permanently remove the data from the cars table. Only the authorized user of admin can delete car data. So anyone who doesn't have the role of admin will not be able to delete any car data. After the deletion process occurs the code will redirect the user back to the list page. But if the user was in a filtered/sorted search they will go back to that filtered/sorted search and they wont have to filter search again.

Saved: 11/24/2025 10:45:43 PM

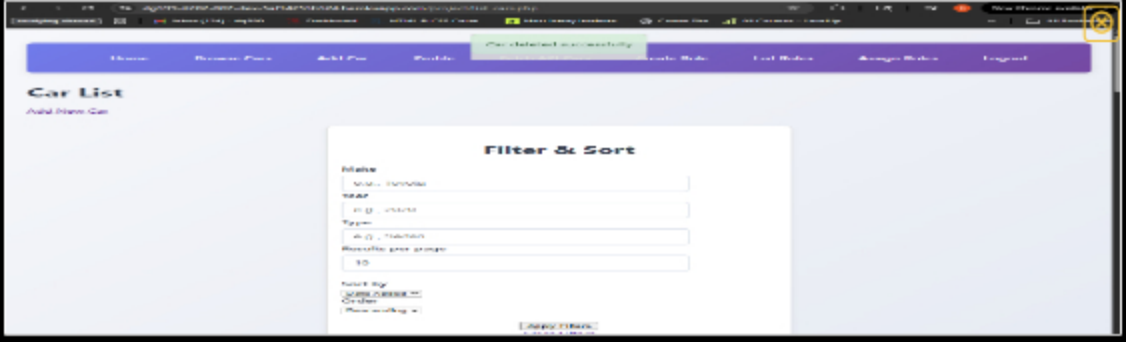
Task #2 (60 pts.) - Examples

Progress: 100%

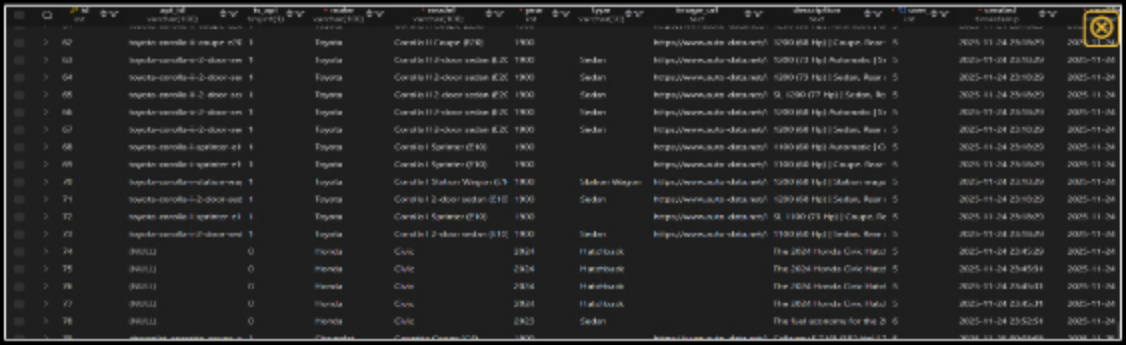
Details:

- Show a successful delete message
- Show a before and after database view of the record being deleted (clearly caption)
- Successful update should have an appropriate message shown
- The updated data should be shown in the form
- Any errors should have user-friendly messages shown

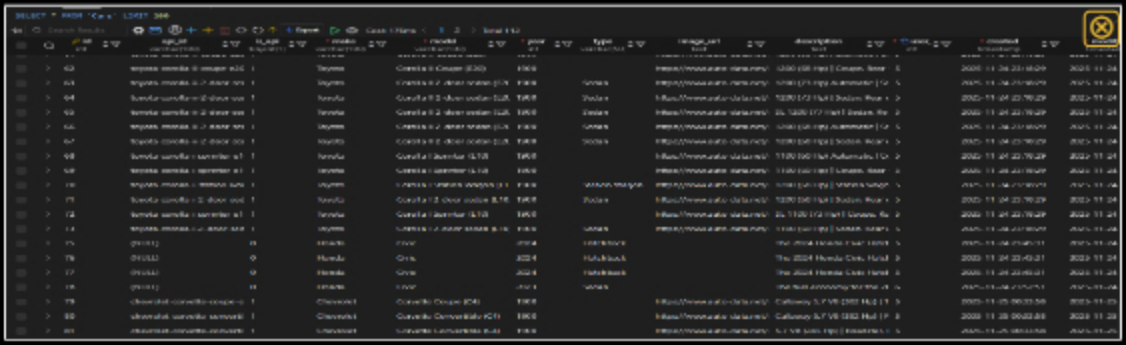
delete js validation



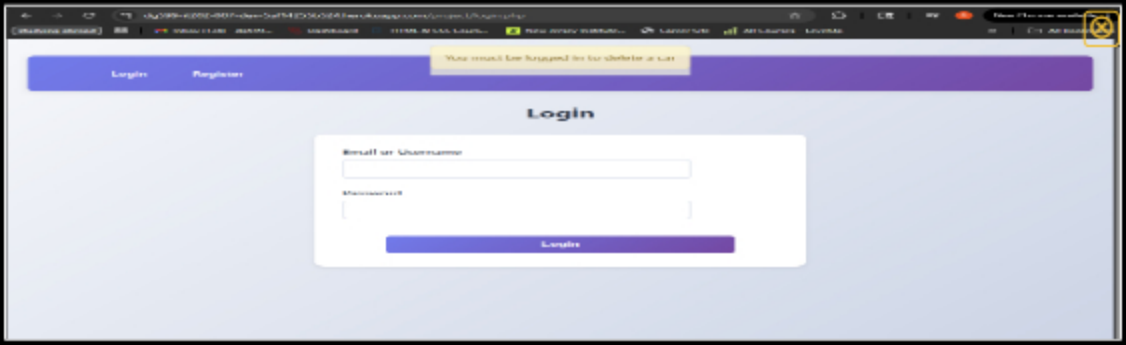
successful delete



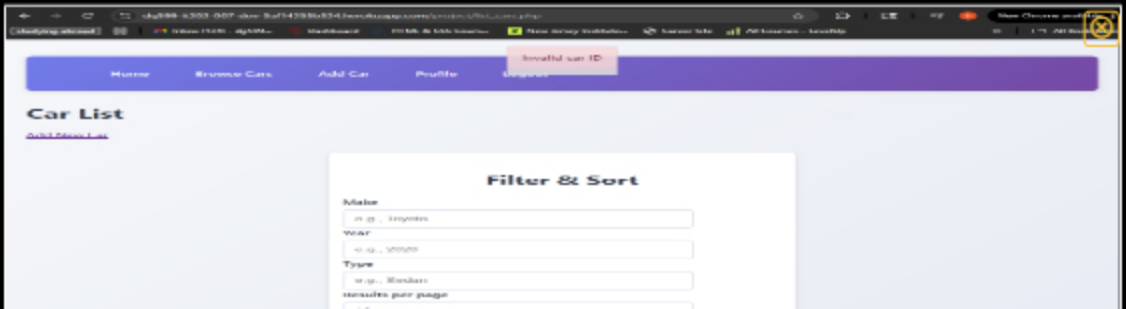
before delete table



after delete table



delete validation



delete validation p2

Saved: 11/24/2025 10:55:09 PM

Task #3 (30 pts.) - Links

Progress: 100%

Details:

- Include the heroku prod link to this page (even though it's not a full page, it's more like how logout works)
- Include pull request link for this feature

URL #1
https://dg599-it202-007-dev-5af14255b524.herokuapp.com/project/delete_car.php

URL
https://dg599-it202-007-dev-

URL #2
https://github.com/dylangarca/dg599-IT202-007/pull/41

URL
https://github.com/dylangarc

Saved: 11/24/2025 10:57:26 PM

Section #7: (50 pts.) Misc

Progress: 100%

Task #1 (16.67 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

Details:
From the Commits tab of the Pull Request screenshot the commit history

commit page

Saved: 11/24/2025 10:59:07 PM

Part 2:

Progress: 100%

Details:
Include the link to the Pull Request (should end in `/pull/#`)

URL #1
https://github.com/dylangarca/dg599-IT202-007/pull/42

URL
https://github.com/dylangarca/dg

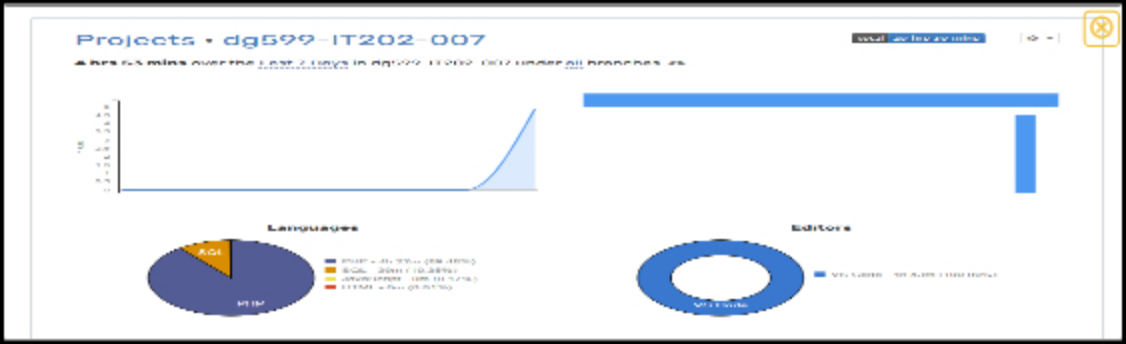
Saved: 11/24/2025 10:59:07 PM

Task #2 (16.67 pts.) - WakaTime - Activity

Progress: 100%

Details:

- Visit the WakaTime.com Dashboard
- Click `Projects` and find your repository
- Capture the overall time at the top that includes the repository name
- Capture the individual time at the bottom that includes the file time
- Note: The duration isn't relevant for the grade and the visual graphs aren't necessary



wakatime

wakatime

Saved: 11/24/2025 11:00:30 PM

Task #3 (16.67 pts.) - Reflection

Progress: 100%

Task #1 (0.33 pts.) - What did you learn?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

I would say that I learned alot with milestone 2. I think the biggest thing would have to be the api integration with the database. I learned how to desing database tables properly to store api data with the proper columns. I had to think which fields i was going to use from the api and than extract the specific data i was going to use. I also learned about the CRUD operations and how to implement them in webpages.

Saved: 11/24/2025 11:14:33 PM

Task #2 (0.33 pts.) - What was the easiest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

I would say that the easiest part of milestone 2 was the create page because it was really similar to the milestone 1 files. It involved many similar things like the 3 layers of validation and the handle submissions and error messages. Since i knew how to do this from milestone 1 that is why it was on the easier side of milestone 2. It was also a straight foward process and it had clear requirements.

Saved: 11/24/2025 11:06:19 PM

Task #3 (0.33 pts.) - What was the hardest part of the assignment?

Progress: 100%

Details:
Briefly answer the question (at least a few decent sentences)

Your Response:

I would say the hardest part of milestone 2 was the api integration with the duplicate handling. The reason for this was i encountered an error with the response structure of the api. So at first i was looking for the data in a data array while it was actually in a results array. Than after I fixed that issue i had to parse the title to get the model and make information. I also encountered a problem with the api becuase until the api integration part i didn't realize that this api does not have the year of the vehicles so i had to first check if the results array had the year of the car and if not set the default to the oldest year.

Saved: 11/24/2025 11:11:44 PM