

# Submission Worksheet

**CLICK TO GRADE**

<https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-api-project-milestone-2-2024/grade/df39>

IT202-008-S2024 - [IT202] API Project Milestone 2 2024

## Submissions:

Submission Selection

1 Submission [active] 4/19/2024 3:08:33 PM

## Instructions

[▲ COLLAPSE ▲](#)

Implement the Milestone 2 features from the project's proposal

document: <https://docs.google.com/document/d/1XE96a8DQ52Vp49XACBDTNCq0xYDt3kF29cO88E>

Make sure you add your ucid/date as code comments where code changes are done

All code changes should reach the Milestone2 branch

Create a pull request from Milestone2 to dev and keep it open until you get the output PDF from this assignment.

Gather the evidence of feature completion based on the below tasks.

Once finished, get the output PDF and copy/move it to your repository folder on your local machine.

Run the necessary git add, commit, and push steps to move it to GitHub

Complete the pull request that was opened earlier

Create and merge a pull request from dev to prod

Upload the same output PDF to Canvas

Branch name: Milestone2

Tasks: 29 Points: 10.00

 Define Appropriate Tables for Data (1 pt.)

[▲ COLLAPSE ▲](#)

 Task #1 - Points: 1

Text: Screenshots of Table SQL

Checklist

\*The checkboxes are for your own tracking

| #    | Points | Details   |
|------|--------|---|
| ■ #1 | 1      | Table(s) should have the 3 core columns we'll always be using (id, created, modified) plus additional columns for the incoming API data |
| ■ #2 | 1      | Columns should be logical and thought out (not valid to have a single field of JSON data or similar)                                    |
| ■ #3 | 1      | Clearly caption screenshots   |

## Task Screenshots:

### Gallery Style: Large View

**Small              Medium              Large**

public\_html > Project > sql > 006\_create\_table\_movies.sql > ...

Run | New Tab | Copy | Active Connection

```
1 CREATE TABLE IF NOT EXISTS `Movies` [
2     `id` INT NOT NULL AUTO_INCREMENT,
3     `created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
4     `modified` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
5     `title` VARCHAR(255) NOT NULL,
6     `year` YEAR NOT NULL,
7     `imdb_id` VARCHAR(50) NOT NULL,
8     `source` VARCHAR(255) NOT NULL,
9     PRIMARY KEY (`id`),
10    UNIQUE (`title`)
11    -- DF39 4/19/2024
12 ]
```

3 Core columns and additional columns for API data (title, year, source, imdb\_id)

### **Checklist Items (1)**

#1 Table(s) should have the 3 core columns we'll always be using (`id`, `created`, `modified`) plus additional columns for the incoming API data

| Tables (4) |            | Search results |                     |                     |   |      |                            |        |  |  |  |
|------------|------------|----------------|---------------------|---------------------|---|------|----------------------------|--------|--|--|--|
|            |            | id             | created timestamp   | modified timestamp  | title                                     | year | imdb_id                    | source |  |  |  |
| >          | Movies     | 1              | 2024-04-19 16:46:48 | 2024-04-19 16:46:48 | Harry Potter and the Goblet of Fire       | 2005 | tt0390373                  | api    |  |  |  |
| >          | UserRoles  | 2              | 2024-04-19 16:46:48 | 2024-04-19 16:46:48 | Harry Potter and the Half-Blood Prince    | 2009 | tt0417741                  | api    |  |  |  |
| >          | Users      | 3              | 2024-04-19 16:46:48 | 2024-04-19 16:46:48 | Harry Potter and the Order of the Phoenix | 2007 | tt0373889                  | api    |  |  |  |
| >          | Roles      | 4              | 2024-04-19 16:46:48 | 2024-04-19 16:46:48 | Harry Potter and the Prisoner of Azkaban  | 2001 | tt0241527                  | api    |  |  |  |
| >          | Views      | 5              | 2024-04-19 16:46:49 | 2024-04-19 16:46:49 | Harry Potter and the Prisoner of Azkaban  | 2004 | tt0304141                  | api    |  |  |  |
| >          | Procedures | 6              | 2024-04-19 16:46:49 | 2024-04-19 16:46:49 | Harry Potter and the Sorcerer's Stone     | 2001 | tt0241527,tt1201607,tt0380 | api    |  |  |  |
| >          | Functions  | 7              | 2024-04-19 16:46:49 | 2024-04-19 16:46:49 | Harry Potter: A History Of Magic          | 2017 | tt7783322                  | api    |  |  |  |

|      |     |                     |                     |                                |      |            |     |
|------|-----|---------------------|---------------------|--------------------------------|------|------------|-----|
| > 8  | 241 | 2024-04-19 16:46:49 | 2024-04-19 16:46:49 | Harry Potter: The Making of D  | 2014 | tt5901980  | api |
| > 9  | 242 | 2024-04-19 16:46:49 | 2024-04-19 16:46:49 | Midnight Screenings Harry Po   | 2011 | tt3116138  | api |
| > 10 | 243 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | 365 Days of Happiness          | 2011 | tt2094748  | api |
| > 11 | 244 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | 42 Seconds Of Happiness        | 2016 | tt3026124  | api |
| > 12 | 245 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | A Dose of Happiness            | 2019 | tt10249780 | api |
| > 13 | 246 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | A Journey Of Happiness         | 2019 | tt9809728  | api |
| > 14 | 247 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | A Moment of Happiness          | 2020 | tt11674528 | api |
| > 15 | 248 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | A Tale of Happiness            | 1988 | tt0096350  | api |
| > 16 | 249 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | Agent of Happiness             | 2024 | tt21224854 | api |
| > 17 | 250 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | Amoritism: Happiness from t    | 1935 | tt0026609  | api |
| > 18 | 251 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | And the Pursuit of Happiness   | 1988 | tt0092581  | api |
| > 19 | 252 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | Arnold Cautionous and the Happ | 2023 | tt21105818 | api |
| > 20 | 253 | 2024-04-19 16:46:56 | 2024-04-19 16:46:56 | BET Presents Love & Happines   | 2016 | tt6269718  | api |

## The actual database with my API information

### Checklist Items (2)

#2 Columns should be logical and thought out (not valid to have a single field of JSON data or similar)

#3 Clearly caption screenshots

#### Task #2 - Points: 1

Text: Explain the design

### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Note the different fields and their purpose  |
| <input checked="" type="checkbox"/> #2 | 1      | Note if you needed to normalize the data into separate tables or if one table fit your needs |

### Response:

I only needed one table for my design due to just needing movie titles and their appropriate data with it (year, title, source, imdb\_id). All of this information is coming from my API which is based on the imdb website. Title is obviously the title of the movie, year is the year the movie was released, source is if the data is being inserted from manual entry or coming in the form the API and imdb id is the id for finding the movie in the imdb website.

#### Task #3 - Points: 1

Text: Add the pull request link for the branch related to this feature

### Details:

Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature.

● Data Creation Page (2 pts.)

[COLLAPSE](#)

● Task #1 - Points: 1

Text: Screenshots of the creation page

[COLLAPSE](#)

Checklist

\*The checkboxes are for your own tracking

| #                           | Points | Details  |
|-----------------------------|--------|--|
| <input type="checkbox"/> #1 | 1      | Show potentially valid data filled in for the custom creation page   |
| <input type="checkbox"/> #2 | 1      | Show how the API data is fetched for API data (must be server-side)  |
| <input type="checkbox"/> #3 | 1      | Show examples of validation messages                                 |
| <input type="checkbox"/> #4 | 1      | Show an example of successful creation message                       |
| <input type="checkbox"/> #5 | 1      | Design/Style should be considered (i.e., bootstrap, custom css, etc) |
| <input type="checkbox"/> #6 | 1      | Make sure the heroku dev url is visible in the address bar           |
| <input type="checkbox"/> #7 | 1      | Clearly caption screenshots  |

Task Screenshots:

Gallery Style: Large View

Small

Medium

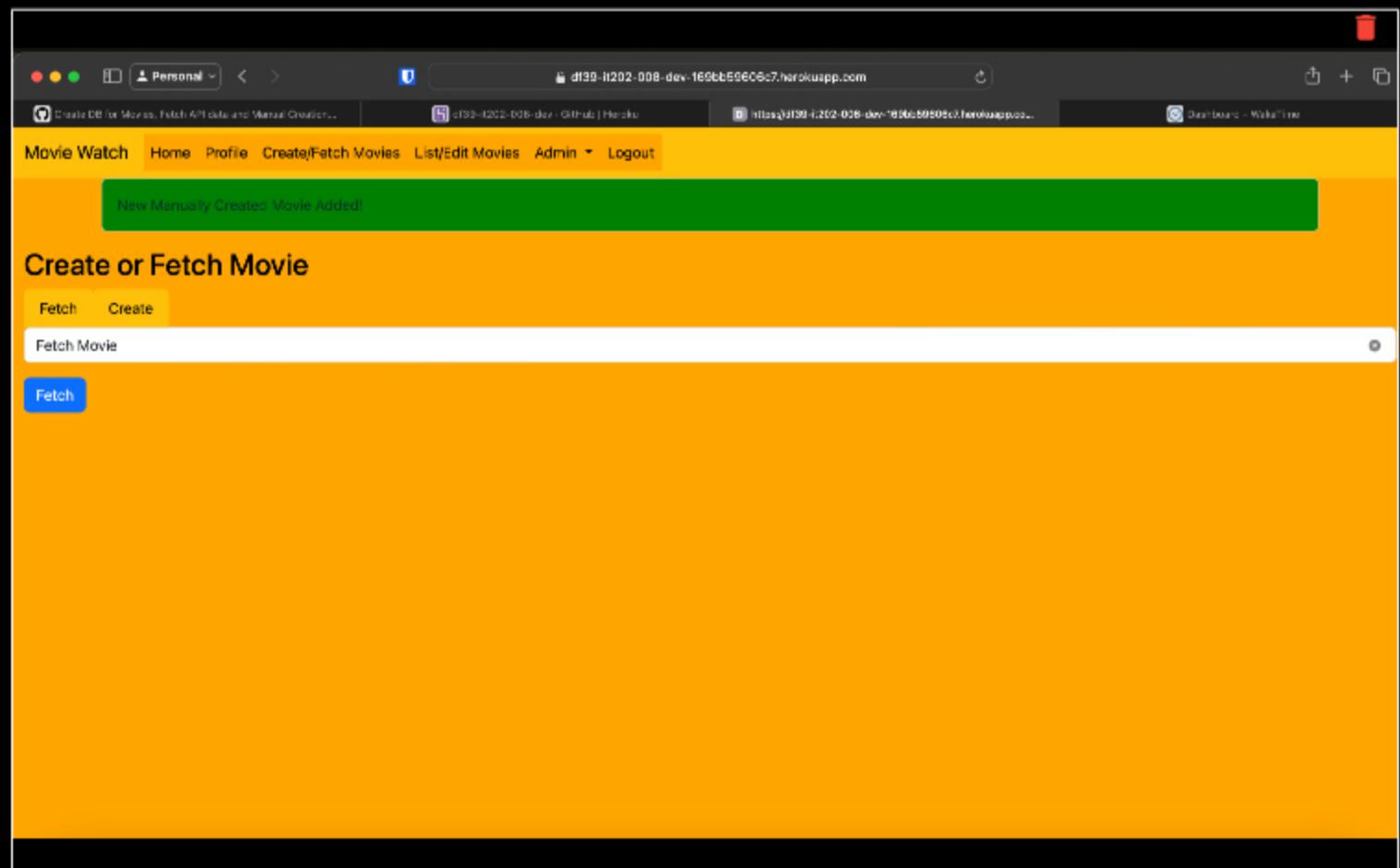
Large

The screenshot shows a web browser window with the following details:

- Title Bar:** df39-it202-008-dev-169bb59606c7.herokuapp.com
- Address Bar:** df39-it202-008-dev-169bb59606c7.herokuapp.com
- Page Content:**
  - Header:** Create or Fetch Movie
  - Buttons:** Fetch, Create
  - Form Fields:**
    - Movie Name: IT202 The Movie
    - Movie Year: 2024
    - IMDB ID: tt3948
  - Buttons:** Create

## Checklist Items (1)

### #1 Show potentially valid data filled in for the custom creation page



This screenshot shows the bootstrap styling with a successful manual entry being made message. The heroku dev url is in the url bar

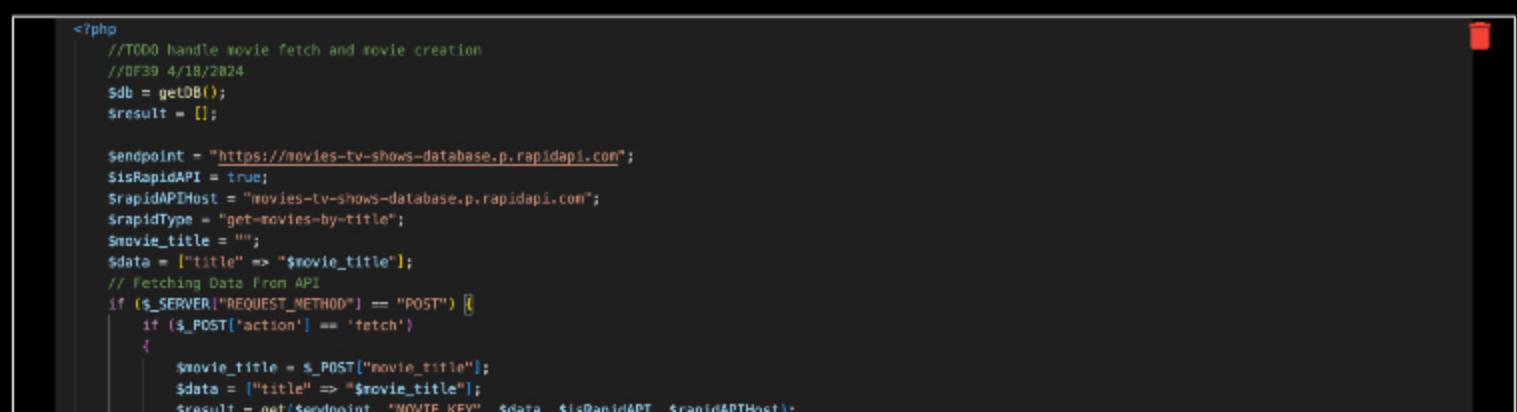
## Checklist Items (4)

### #4 Show an example of successful creation message

### #5 Design/Style should be considered (i.e., bootstrap, custom css, etc)

### #6 Make sure the heroku dev url is visible in the address bar

### #7 Clearly caption screenshots



```

    }
    // Getting Data From Manual Creation Page
    elseif ($_POST['action'] == 'create') {
        try {
            // Manual Entry Insert into DB
            $stmt = $db->prepare("INSERT INTO Movies (year, imdb_id, title, source) VALUES(:year, :imdb_id, :title, :source)");
            $stmt->execute([
                ':year' => $_POST['year'],
                ':imdb_id' => $_POST['imdb_id'],
                ':title' => $_POST['title'],
                ':source' => 'manual'
            ]);
            flash('New Manually Created Movie Added!', "success");
        } catch (PDOException $e) {
            if ($e->errorInfo[1] == 1062) { // 1062 is the SQLSTATE for a unique constraint violation
                flash('A movie with the same title already exists', "warning");
            } else {
                throw $e; // rethrow the exception if it's not a unique constraint violation
            }
        }
    }
}
else {
    $result = [];
}

```

This screenshot shows the API data being fetched and assigned to the \$result variable.

## Checklist Items (1)

### #2 Show how the API data is fetched for API data (must be server-side)

The screenshot shows a browser window with the URL <https://d39-k222-008-dev-1696b69609c7.herokuapp.com/>. The page title is "Movie Watch". The main content area has a red background with white text: "All fields must be filled out". Below this, there is a heading "Create or Fetch Movie" and two tabs: "Fetch" and "Create" (which is selected). There are four input fields: "Movie Name" (placeholder "Movie Name"), "Movie Year" (placeholder "Movie Year"), "IMDB ID" (placeholder "Movie IMDB ID"), and a "Movie IMDB ID" field which is empty. At the bottom of the page, there is a "Create" button.

At the bottom of the browser window, the developer tools are open, specifically the Elements tab. It shows the DOM structure of the "Create" button. The element is a `<button type="submit" id="createForm" onsubmit="return validateForm('createForm');">Create`. The developer tools also show the computed styles for the button, including a border and padding.

This screenshot shows validation message for create movie page and that the fields must not be empty.

## Checklist Items (1)

### #3 Show examples of validation messages

The screenshot shows a browser window with a yellow background at the bottom. A red error message box is displayed with the text "Invalid Data Type for Year, Movie or IMDB\_ID".

## Create or Fetch Movie

Patch Create

Movie Name

Movie Title

Movie Year

cknscj

IMDB ID

tt097897

The screenshot shows a browser's developer tools with the "Elements" tab selected. A modal dialog is open, and the "Console" tab shows validation errors:

```
Uncaught Error: Validation failed for element movie_year
    at https://d38-t202-009-dev-169bb59506c7.herokuapp.com/movies/create?_method=POST
```

The "Elements" panel shows the DOM structure of the modal, including the input field for "movie\_year". The "Style" panel shows the CSS properties for the input field.

This screenshot shows validation in the create page of incorrect data types being inputted (this case it is the year not being a number)

### Checklist Items (1)

#### #3 Show examples of validation messages

The screenshot shows a browser's developer tools with the "Elements" tab selected. A modal dialog is open, and the "Console" tab shows validation errors:

```
Uncaught Error: Validation failed for element movie_title
    at https://d38-t202-009-dev-169bb59506c7.herokuapp.com/movies/create?_method=POST
```

The "Elements" panel shows the DOM structure of the modal, including the input field for "movie\_title". The "Style" panel shows the CSS properties for the input field.

This screenshot shows validation of the fetch page and the input filed being blank.

### Checklist Items (1)

## #3 Show examples of validation messages

## Task #2 - Points: 1

Text: Screenshots of creation page code

## Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Form should have correct data types for each property being requested  |
| <input checked="" type="checkbox"/> #2 | 1      | Form should have correct validation for each field (HTML, JS, and PHP) |
| <input checked="" type="checkbox"/> #3 | 1      | Successful creation should have a user-friendly message                |
| <input checked="" type="checkbox"/> #4 | 1      | Any errors should have user-friendly messages                          |
| <input checked="" type="checkbox"/> #5 | 1      | Include the form/process for fetching API data                         |
| <input checked="" type="checkbox"/> #6 | 1      | Include some indicator between custom data and API data                |
| <input checked="" type="checkbox"/> #7 | 1      | Include any other rules like role guards and login checks              |
| <input checked="" type="checkbox"/> #8 | 1      | Include ucid/date comments for each code screenshot                    |
| <input checked="" type="checkbox"/> #9 | 1      | Clearly caption screenshots  |

## Task Screenshots:

## Gallery Style: Large View

|       |        |       |
|-------|--------|-------|
| Small | Medium | Large |
|-------|--------|-------|

```
sublime_text /Project 2/admin 2> ./create_movie.php > ...
1  <?php
2  //note we need to go up 2 more directory
3  //00:39 4/19/2018
4  require __DIR__ . "/../../../../partials/nav.php";
5
6  ?>
7
8
9  <?php if (isset($results)) : ?>
10  <?php foreach ($results as $result) : ?>
11      <?php
12          <?php var_export($stack); ?>
13      </?php
14      <?php endforeach; ?>
15  <?php endif; ?>
16
17  <div class="container">
18      <?php CreateOrFetchMovie(); ?>
19      <?php if ($stack) : ?>
20          <ul class="nav-justified">
21              <a class="nav-link bg-warning" href="#" onClick="switchTab('create')"><=Fetch</a>
22          </a>
23          <li class="nav-item">
24              <a class="nav-link bg-warning" href="#" onClick="switchTab('fetch')"><=Creates</a>
25          </li>
26      </ul>
27  </div>
28  <div id="fetch" class="tab-target">
29      <form method="POST" id="fetchForm" onSubmit="return validateForm('fetchForm');">
30          <?php render_input(['type' => 'hidden', 'name' => 'Movie Title', 'placeholder' => 'Movie Title', 'value' => 'Fetch Movie', 'rules' => ['required' => 'required']] ); ?>
31          <?php render_input(['type' => 'hidden', 'name' => 'Action', 'value' => 'Fetch'] ); ?>
32          <?php render_button(['name' => 'Fetch', 'type' => 'submit']) ; ?>
33      </form>
34  </div>
35  <div id="create" style="display: none;" class="tab-target">
36      <form method="POST" id="createForm" onSubmit="return validateForm('createForm');">
37          <?php render_input(['type' => 'text', 'name' => 'title', 'label' => 'Movie Name', 'placeholder' => 'Movie Name', 'value' => 'Movie Title', 'rules' => ['required' => 'required']] ); ?>
38          <?php render_input(['type' => 'text', 'name' => 'year', 'label' => 'Movie Year', 'placeholder' => 'Movie Year', 'value' => 'Year of Release', 'rules' => ['required' => 'required']] ); ?>
39          <?php render_input(['type' => 'text', 'name' => 'id', 'label' => 'Movie ID', 'placeholder' => 'Movie ID', 'value' => '0000 00', 'rules' => ['required' => 'required']] ); ?>
40
41          <?php render_input(['type' => 'hidden', 'name' => 'Action', 'value' => 'Create'] ); ?>
42          <?php render_button(['text' => 'Create', 'type' => 'submit', 'text' => 'Create'] ); ?>
43      </form>
44  </div>
45  <div class="row">
46      <div>
47          <script>
48              function switchTab(tab)
49              {
50                  let target = document.getElementById(tab);
51                  if (target)
52                  {
53                      let elas = document.getElementsByClassName('tab-target');
54                      for (let ele of elas)
55                      {
56                          ele.style.display = ele.id === tab ? "block" : "none";
57                      }
58                  }
59              }
60          </script>
61      </div>
62  </div>
```

This caption shows all of the forms correct data types for data being fetched from api and created by the user. The only one not shown is the source data (will include in next screenshot) As well as HTML validation is shown in this screenshot of each field being required.

### Checklist Items (1)

#1 Form should have correct data types for each property being requested

This screenshot shows the user's title entry for the movie title being passed as a parameter in the query from the API. It also shows a flash message with a successful creation and a flash message if the movie title already exists (duplicate checking). This screenshot also shows the form values for year, id and title being passed into the manual entry into the db for manual creation. It also includes a differentiation between API and Manual entries being passed into the source column in the db. I have no rules for roles or login checks.

### **Checklist Items (4)**

#3 Successful creation should have a user-friendly message

#### #4 Any errors should have user-friendly messages

## #5 Include the form/process for fetching API data

#6 Include some indicator between custom data and API data

```
<script>
    // DF39 4/19/2024
    function validateForm(formId) {
        let form = document.getElementById(formId)
```

```

let title = form['title'];
let year = form['year'];
let imbd_id = form['imdb_id'];
let movie_title: any [];

let movie_title = form['movie_title'];

if (formId === 'createForm') {
    if (!title.value || !year.value || !imbd_id.value) {
        flash("All fields must be filled out");
        return false; // Stop the form from submitting
    }

    // Validating Data Types
    if (!isNaN(title.value) || isNaN(year.value) || isNaN(imbd_id.value)) {
        flash("Invalid Data Type for Year, Movie or IMDB_ID", "warning");
        return false;
    }
} else if (formId === 'fetchForm') {
    if (!movie_title.value) {
        flash("Movie title must be filled out");
        return false; // Stop the form from submitting
    }
}

return true;
}
</script>

```

This JavaScript validate function will check if the input fields from the user are the correct data types being inputted.

### Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

### Task #3 - Points: 1

Text: Screenshot of records from DB

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Show at least one record fetched from the API          |
| <input checked="" type="checkbox"/> #2 | 1      | Show at least one record created via the creation form |
| <input checked="" type="checkbox"/> #3 | 1      | Note what differs                                      |
| <input checked="" type="checkbox"/> #4 | 1      | Clearly caption screenshots                            |

#### Task Screenshots:

##### Gallery Style: Large View

Small      Medium      Large



[Fetch](#)

Fetching Toy Story 1 from the api being shown in this and next screenshot.

### Checklist Items (1)

#1 Show at least one record fetched from the API

The screenshot shows a web browser window with the address bar containing "df39-it202-008-dev-109bb59606c7.herokuapp.com". The page title is "Create or Fetch Movie". The form has two tabs: "Fetch" (selected) and "Create". The "Movie Name" field contains "IT202 The Movie". The "Movie Year" field contains "2024". The "IMDB ID" field contains "tt1234567". A blue "Create" button is visible at the bottom left of the form area.

This screen shows me creating a manual entry for the "IT202 The Movie" movie record to be added into the db.

### Checklist Items (2)

#2 Show at least one record created via the creation form

#4 Clearly caption screenshots

The screenshot shows a MySQL database interface with the following details:

- Schemas:** IT202-008 8.0.35-0u...
- Tables:** Movies (selected), Roles, UserRoles, Users, Views, Procedures, Functions, information\_schema.
- Properties:** DATA, Monitor
- Query:** SELECT \* FROM 'Movies' LIMIT 100
- Table Structure:**

|  | Q   | id  | created timestamp   | modified timestamp  | title                      | year | imdb_id   | source |
|--|-----|-----|---------------------|---------------------|----------------------------|------|-----------|--------|
|  | > 1 | 522 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Charlie: A Toy Story       | 2013 | tt2475692 | api    |
|  | > 2 | 523 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Making Toy Story           | 1996 | tt0245266 | api    |
|  | > 3 | 524 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 2                | 1999 | tt0120363 | api    |
|  | > 4 | 525 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 3                | 2010 | tt0435761 | api    |
|  | > 5 | 526 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 4                | 2019 | tt1879376 | api    |
|  | > 6 | 527 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story of Terror!       | 2013 | tt2446040 | api    |
|  | > 7 | 528 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story That Time Forgot | 2014 | tt3473654 | api    |
|  | > 8 | 529 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story                  | 1995 | tt0114709 | api    |
|  | > 9 | 530 | 2024-04-19 19:48:57 | 2024-04-19 19:48:57 | IT202 The Movie            | 2024 | tt1234567 | manual |

This screenshot shows the manual creation of the IT202 movie being created by the user and being differentiated by the source column being "manual". This screenshot also shows the Toy Story movie information being fetched from the API and it showing that in the source column saying 'api'

#### Checklist Items (4)

#1 Show at least one record fetched from the API

#2 Show at least one record created via the creation form

#3 Note what differs

#4 Clearly caption screenshots

#### Task #4 - Points: 1

Text: Explain the process

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details   |
|--|--------|---|
| <input checked="" type="checkbox"/> #1 | 1      | Provide a high-level step-by-step of how fetching API data works and gets added to your DB (include how duplicates are handled) |
| <input checked="" type="checkbox"/> #2 | 1      | Provide a high-level step-by-step of how creating custom data works and gets added to   |

|  |   |  |
|--|---|--|
| <input checked="" type="checkbox"/> #2 | 1 | your DB (include how duplicates are handled)               |
| <input checked="" type="checkbox"/> #3 | 1 | Briefly describe the validations for the applicable fields |
| <input checked="" type="checkbox"/> #4 | 1 | Describe how duplicate data is handled                     |

#### Response:

##### API Data being Fetched and Inserted into DB, duplicate handling and form validation:

So first a fetch form is shown to the user to search a movie title from the API based on what the user inputs this is taken and assigned to a variable from the form value being `movie_name`. This value from the user is then passed into the query of API by assigning the `movie_title` value from the form into the `$data` array and assigned the `title` key in this array. The server will then ensure that it will hit the endpoint of the API for data if the form has been submitted and if the value of `action` is equal to the value of the fetch form which is 'fetch'. It then queries the database using the get command and assigns the result array to the variable `$results`. The get command essentially has the url for the endpoint of the api, the `$data` array which has the movie title from the user, `$isRapidAPI` which is a boolean that runs true if it is coming from rapid api and if it is it will use specific headers from the api. It also contains the `movie_key` which is the api key from my specific account in rapid api. Once the results from the API is assigned to the `$result` variable it will then loop over the results of this associative array and input the results into the db. So it will assign the result of the year, `imdb_id` and title from the API to the movies db. In this same execute statement because this information is coming from the API, the source column in the db will also getting assigned the 'api' name to know where this information came from. Duplicate data is handled before the results are passed into the DB. The title column in the DB is a unique constraint. So before entering the information into the db, there is a try catch block that will attempt to insert the title into the db and if a error 1062 is thrown (this is a unique constraint error) a flash message will appear to the user that a movie with this title already exists and the information will not get added to the db. As far as validation in the form the user input field of title is assigned a value title and if this field is empty a javascript function called validate form will check of this field is empty and if it is it will send the user a flash message saying that the input field must not be empty.

##### User Input being Inserted into DB for Manual entry with duplicate handling and validation:

The create form includes the year, title and `imdb_id` field for the user to input. These input fields are assigned to variables that will be checked by the java script validate form function that will check if the data types are not correct, specifically for the year. The year input field must be a number and if the user does not use a number a flash message will appear to them telling them that the year, `imdb_id` or title has the incorrect data type being passed. Once the correct data types are inserted these input fields are assigned to values and passed into a sql execute statement statement. This statement is in a try catch block as well to ensure duplicate data is not inserted. Due to the title being a unique constraint in the table if the user attempts to enter a duplicate title that is in the data base, the try catch block will catch this and catch a 1062 error which will flash a message to the user that this title is already in the data base. If the try catch block does not catch this error then the data will be passed to the correct columns in the db and get inserted.

#### Task #5 - Points: 1

Text: Add related links

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details   |
|--|--------|---|
| <input checked="" type="checkbox"/> #1 | 1      | Include the heroku prod link for this page  |
| <input checked="" type="checkbox"/> #2 | 1      | Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature |

URL #1

<https://github.com/dunnfall/df39-it202-008/pull/53>

URL #2

[https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/list\\_movies.php](https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/list_movies.php)

● Data List Page (many entities) (2 pts.)

[COLLAPSE](#)

● Task #1 - Points: 1

Text: Screenshots of the list page

Checklist

\*The checkboxes are for your own tracking

| #    | Points | Details   |
|------|--------|---|
| ■ #1 | 1      | Show the page of your entities listed (have a reasonable number shown)  |
| ■ #2 | 1      | Show the filter/sort form based on your data and the required limit field   |
| ■ #3 | 1      | Demonstrate a few varied filters/sorts  |
| ■ #4 | 1      | Demonstrate a filter that doesn't have any records (should show an appropriate message)   |
| ■ #5 | 1      | Each list item should have a link of single view (i.e., details), edit, and delete (some of which may only be visible to admin users) |
| ■ #6 | 1      | Each list item should have a summary of the entity (likely won't be the entire entity data)   |
| ■ #7 | 1      | Design/Style should be considered (i.e., bootstrap, custom css, etc)  |
| ■ #8 | 1      | Make sure the heroku dev url is visible in the address bar  |
| ■ #9 | 1      | Clearly caption screenshots   |

Task Screenshots:

Gallery Style: Large View

Small

Medium

Large

| id  | title                | year | Actions  |
|-----|----------------------|------|--|
| 522 | Charlie: A Toy Story | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 623 | Making Toy Story     | 1995 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 624 | Toy Story 2          | 1999 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 525 | Toy Story 3          | 2010 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 628 | Toy Story 4          | 2019 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |

|     |                            |      |                      |                      |                        |
|-----|----------------------------|------|----------------------|----------------------|------------------------|
| 527 | Toy Story of Terror!       | 2013 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 528 | Toy Story That Time Forgot | 2014 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 529 | Toy Story                  | 1995 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 530 | IT202 The Movie            | 2024 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 531 | Captain Scarface           | 1953 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 532 | Scarface                   | 1983 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |

This screenshot shows the page of listed movies with a few movies for examples. It also shows a view/edit/delete button listed next to each movie. It also shows the dev url in the url bar from heroku. This page also has bootstrap styling and the filter/sort feature buttons and fields at the top.

### Checklist Items (5)

#1 Show the page of your entities listed (have a reasonable number shown)

#2 Show the filter/sort form based on your data and the required limit field

#7 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#8 Make sure the heroku dev url is visible in the address bar

#9 Clearly caption screenshots

The screenshot shows a web application interface for managing movies. At the top, there's a navigation bar with links: Movie Watch, Home, Profile, Create/Fetch Movies, List/Edit Movies, Admin, and Logout. Below the navigation is a yellow header bar containing the title "List Movies". Underneath the header, there's a search/filter form with fields for "Title" (containing "Lion King"), "Year" (containing "Year"), "Sort" (set to "Title"), "Order" (set to "ASC"), and "Limit" (set to "10"). There are also "Filter" and "Clear" buttons. Below the form is a section titled "Latest Movies" which displays a table of movie records. The table columns are "Id", "title", "year", and "Actions". The data in the table is as follows:

| Id  | title                                 | year | Actions  |
|-----|---------------------------------------|------|--|
| 538 | Edward VIII: The Lion King            | 2024 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 539 | PWFG: Rear of The Lion Kingdom Part 1 | 1992 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 542 | The Lion King                         | 1994 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 540 | The Lion King 1½                      | 2004 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 541 | The Lion King II: Simba's Pride       | 1998 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 544 | The Making of the Lion King           | 1994 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |

This screenshot shows a sort by title where the title being sorted is lion king and the results are only lion king movies in ascending alphabetical order.

### Checklist Items (1)

## Checklist Items (1)

### #2 Show the filter/sort form based on your data and the required limit field

The screenshot shows a web browser window with a yellow header bar. The header bar contains several tabs and links, including "Movie Watch", "Home", "Profile", "Create/Fetch Movies", "List/Edit Movies", "Admin", and "Logout". Below the header is a search/filter form titled "List Movies". The form has fields for "Title" (containing "Title") and "Year" (containing "1995"). It also includes "Sort" (set to "Title"), "Order" (set to "+"), and "Limit" (set to "10"). Below the form are two buttons: "Filter" (blue) and "Clear" (grey). Underneath the search form is a section titled "Latest Movies" containing a table with two rows of movie data.

| id  | title            | year | Actions   |
|-----|------------------|------|---|
| 523 | Making Toy Story | 1995 | <button>View</button> <button>Edit</button> <button>Delete</button> |
| 529 | Toy Story        | 1995 | <button>View</button> <button>Edit</button> <button>Delete</button> |

This screenshot shows a filter by year and only showing movies made in 1995.

## Checklist Items (1)

### #3 Demonstrate a few varied filters/sorts

The screenshot shows a web browser window with a yellow header bar. The header bar contains several tabs and links, including "Movie Watch", "Home", "Profile", "Create/Fetch Movies", "List/Edit Movies", "Admin", and "Logout". Below the header is a search/filter form titled "List Movies". The form has fields for "Title" (containing "Title") and "Year" (containing "2009"). It also includes "Sort" (set to "Year"), "Order" (set to "-"), and "Limit" (set to "10"). Below the form are two buttons: "Filter" (blue) and "Clear" (grey). Underneath the search form is a section titled "Latest Movies" containing a table with one row of movie data.

| id  | title            | year | Actions   |
|-----|------------------|------|---|
| 523 | Making Toy Story | 1995 | <button>View</button> <button>Edit</button> <button>Delete</button> |

This screenshot shows a filter of movies by the year of 2009 and not having any records shown with an appropriate message saying "No Records to Show"

### Checklist Items (1)

#4 Demonstrate a filter that doesn't have any records (should show an appropriate message)

#### Task #2 - Points: 1

Text: Screenshots of the list page code

#### Checklist

\*The checkboxes are for your own tracking

| #                           | Points | Details   |
|-----------------------------|--------|---|
| <input type="checkbox"/> #1 | 1      | Show the filter/sort form generation  |
| <input type="checkbox"/> #2 | 1      | Show the DB query and how the filter/sort is handled (including the restriction on the limit field) |
| <input type="checkbox"/> #3 | 1      | Show how the output is generated and displayed  |
| <input type="checkbox"/> #4 | 1      | Show any restrictions like role guard or login checks   |
| <input type="checkbox"/> #5 | 1      | Include ucid/date comments for each code screenshot   |
| <input type="checkbox"/> #6 | 1      | Clearly caption screenshots   |

#### Task Screenshots:

##### Gallery Style: Large View

Small

Medium

Large

```
<?php
//note we need to go up 1 more directory
//DF39 4/19/2024
require(__DIR__ . "/../../../../partials/nav.php");
?>

<?php

$form = [
    "type" => "text", "name" => "title", "placeholder" => "Title", "label" => "Title", "include_margin" => false,
    "type" => "text", "name" => "year", "placeholder" => "Year", "label" => "Year", "include_margin" => false,
    "type" => "select", "name" => "sort", "label" => "Sort", "options" => ["title" => "Title", "year" => "Year"], "include_margin" => false,
    "type" => "select", "name" => "order", "label" => "Order", "options" => ["asc" => "+", "desc" => "-"], "include_margin" => false,
    "type" => "number", "name" => "limit", "label" => "Limit", "value" => "10", "include_margin" => false,
];
error_log("Form data: " . var_export($form, true));
```

## This screenshot shows the filter/sort form generation

### Checklist Items (1)

#### #1 Show the filter/sort form generation

```
23 //DF39 4/19/2024
24 $query = "SELECT id, title, year FROM Movies WHERE 1=1";
25 $params = [];
26 $session_key = $_SESSION["session_key"];
27 $id_clear = $_GET["clear"];
28 if ($id_clear) {
29     session_unset($session_key);
30     unset($_GET["clear"]);
31     die(header("location: " . $session_key));
32 } else {
33     $session_data = session_get($session_key);
34 }
35 if (count($_GET) == 0 AND count($session_data) == count($session_keys) > 0) {
36     if ($session_data) {
37         $id = $session_data;
38     }
39 }
40 if (count($_GET) > 0) {
41     session_unset($session_key);
42     $keys = array_keys($_GET);
43     foreach ($form as $k => $v) {
44         if (!array_key_exists($k, $keys)) {
45             $form[$k] = $v;
46         }
47     }
48 }
49 $title;
50 $title = $_GET["title"] ?? false;
51 if (empty($title)) {
52     $query .= " AND title like '%$title%'";
53     $params["title"] = "%$title%";
54 }
55 $year;
56 $year = $_GET["year"] ?? false;
57 if (empty($year)) {
58     $query .= " AND year like '%$year%'";
59     $params["year"] = "%$year%";
60 }
61 $order;
62 $order = $_GET["order"] ?? false;
63 if (!empty($order)) {
64     $query .= " ORDER BY $order";
65 }
66 $start = $_GET["start"] ?? false;
67 if (!empty($start)) {
68     $query .= " LIMIT $start";
69 }
70 $limit = $_GET["limit"] ?? false;
71 if (!empty($limit)) {
72     $query .= " LIMIT $start, $limit";
73 }
74 try {
75     $stmt = $conn->prepare($query, ["IMPLICIT"]);
76     $stmt->execute();
77     $rows = $stmt->fetchAll();
78     $stmt->close();
79     $stmt = null;
80 }
81 catch (Exception $e) {
82     $rows = [];
83 }
84 }
85 //IMPORTANT make sure you fully validate that $start and $order (sql injection possibility)
86 $query .= " ORDER BY start asc";
87 //IMPORTANT make sure you fully validate that $start (sql injection possibility)
88 $query .= " LIMIT $start";
89 }
```

This screen shot shows how the filter/sort is handled and the db query is at the top.

### Checklist Items (1)

#### #2 Show the DB query and how the filter/sort is handled (including the restriction on the limit field)

```
$table = [
    "data" => $results, "title" => "Latest Movies", "ignored columns" => ["id"],
    "edit_url" => get_url("admin/edit_movies.php"),
    "delete_url" => get_url("admin/delete_movies.php"),
    "view_url" => get_url("admin/view_movies.php")
];
?>

<!-- DF39 4/19/2024 -->
<div class="container-fluid">
    <h3>List Movies</h3>
    <form method="GET">
        <div class="row mb-3" style="align-items: flex-end;">
            <?php foreach ($form as $k => $v) : ?>
                <div class="col">
                    <?php render_input($v); ?>
                </div>
            </?php endforeach(); ?>
        </div>
    </form>
    <table border="1" style="width: 100%; border-collapse: collapse;">
        <thead>
            <tr>
                <th>ID</th>
                <th>Title</th>
                <th>Year</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <?php foreach ($results as $row) : ?>
                <tr>
                    <td>$row[id]</td>
                    <td>$row[title]</td>
                    <td>$row[year]</td>
                    <td><a href="$edit_url?id=$row[id]">Edit</a> | <a href="$delete_url?id=$row[id]">Delete</a></td>
                </tr>
            <?php endforeach(); ?>
        </tbody>
    </table>
</div>
```

```

        </div>
        <?php endforeach; ?>

    </div>
    <?php render_button(["text" => "Search", "type" => "submit", "text" => "Filter"]); ?>
    <a href="?clear" class="btn btn-secondary">Clear</a>
</form>
<?php render_table($table); ?>
</div>

<?php
//note we need to go up 1 more directory
require_once(__DIR__ . "/../../../../partials/flash.php");
?>
```

This is how the output of the table and buttons is generated and displayed. I have restrictions admin role restrictions on delete and view movies.

#### Checklist Items (1)

#3 Show how the output is generated and displayed

#### Task #3 - Points: 1

Text: Explain how the page works

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Provide the high-level steps of how the filter/sorting works and how data is displayed |
| <input checked="" type="checkbox"/> #2 | 1      | Summarize what you're showing on the screen  |
| <input checked="" type="checkbox"/> #3 | 1      | Mention your design/style choice   |
| <input checked="" type="checkbox"/> #4 | 1      | Mention which users can interact with the view, edit, and delete links                 |

#### Response:

So a form is created that has multiple fields and each field is shown with an associative array with properties such as type, name, place holder and label. These properties are assigned to values such as year and title. After this an SQL query statement is initialized to pull movies from the movies db. The code then checks if the clear parameter is set int the get array. If it is the session data is deleted and the page reloads without the clear parameter. If the clear parameter isn't set the session data of the script is loaded into the session\_data variable. If the get array is empty and the session data variable isn't the get array gets set to session data. If the get array isn't empty it gets saved to the current session. The code will then loop over the fields in the from array and the appropriate fields of the form get assigned to the get array. The code then checks if the title and year parameters get set in the get array and if they are the query string gets modified with the conditions to the filter of movies by title and year. The sort and order parameters in the get array then are pulled from the get array and verified. If they do end up being correct, the query is gets and order by clause added to it. The query string is executed with the parameters array and the results are stored in the results variable. The table is then displayed using the render\_table function which has the view, edit and delete buttons. The form gets rendered using render\_input function.

#### Task #4 - Points: 1

Text: Add related links

**Checklist**

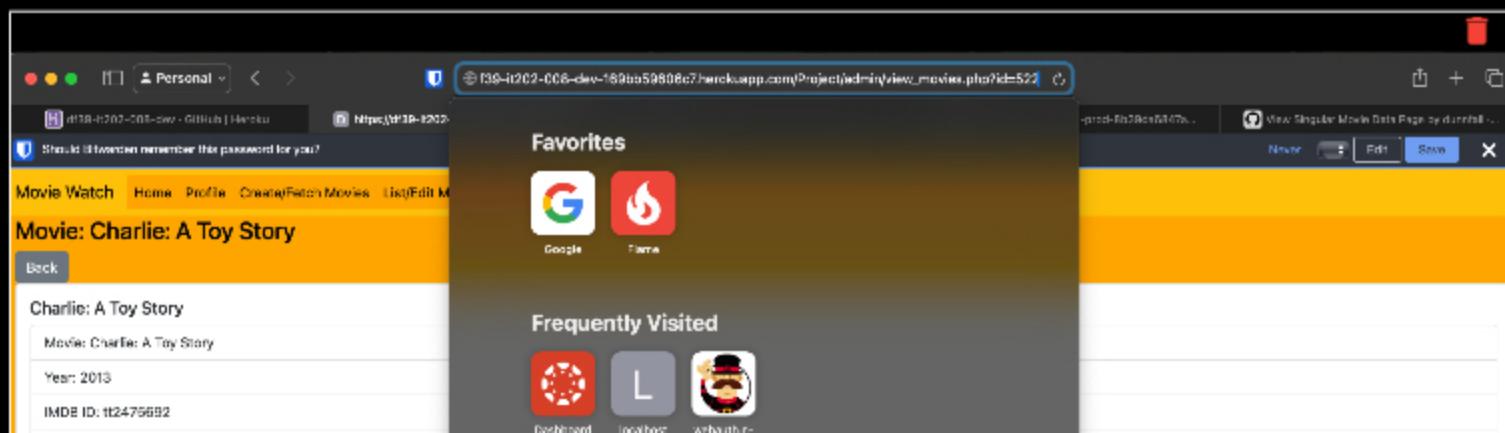
\*The checkboxes are for your own tracking

| #                           | Points | Details   |
|-----------------------------|--------|---|
| <input type="checkbox"/> #1 | 1      | Include the heroku prod link for this page  |
| <input type="checkbox"/> #2 | 1      | Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature |

**URL #1**[https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/list\\_movies.php](https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/list_movies.php)**View Details Page (single entity) (1 pt.)****ACOLLAPSE****Task #1 - Points: 1****Text: Screenshots of the details page****Checklist**

\*The checkboxes are for your own tracking

| #                           | Points | Details   |
|-----------------------------|--------|---|
| <input type="checkbox"/> #1 | 1      | Entity should be fetch by id (via the url)  |
| <input type="checkbox"/> #2 | 1      | A missing id should redirect back to the list page with an applicable message   |
| <input type="checkbox"/> #3 | 1      | Design/Style should be considered (i.e., bootstrap, custom css, etc)  |
| <input type="checkbox"/> #4 | 1      | Data shown should be more detailed/inclusive than the summary view  |
| <input type="checkbox"/> #5 | 1      | There should be a link to edit the entity (this may be an admin-only thing, but it should be present for the respective role)   |
| <input type="checkbox"/> #6 | 1      | There should be a link to delete the entity (this may be an admin-only thing, but it should be present for the respective role) |
| <input type="checkbox"/> #7 | 1      | Make sure the heroku dev url is visible in the address bar  |
| <input type="checkbox"/> #8 | 1      | Clearly caption screenshots   |

**Task Screenshots:****Gallery Style: Large View****Small****Medium****Large**

This screenshot shows a detailed view of a movie record. At the top, there's a header with the title 'Shared with You' and some sharing statistics: '171.6K likes, 892 comments, "this song forever a banger!"'. Below this, there are two cards: one for 'www.tiktok.com/@T1L6JThqf' (1.6M subscribers) and another for 'TikTok - data 4' (30.6K subscribers). At the bottom of the page, there are 'Edit' and 'Delete' buttons.

This screenshot shows the movie being fetched by ID, it shows design/styling with bootstrap. It shows more data being displayed than the list view. List view only contains id/title/year. It also shows a link to edit and delete the entity from the singular view and the heroku dev being show in the URL bar.

## Checklist Items (7)

#1 Entity should be fetch by id (via the url)

#3 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#4 Data shown should be more detailed/inclusive than the summary view

#5 There should be a link to edit the entity (this may be an admin-only thing, but it should be present for the respective role)

#6 There should be a link to delete the entity (this may be an admin-only thing, but it should be present for the respective role)

#7 Make sure the heroku dev url is visible in the address bar

#8 Clearly caption screenshots

This screenshot shows a movie listing interface. At the top, there's a navigation bar with 'Movie Watch' and links for 'Home', 'Profile', 'Create/Edit Movies', 'List/Edit Movies', and 'Logout'. The address bar shows the URL: 'https://d159-h202-008-dev-169bb69606c7.herokuapp.com'. A red error message box says 'Invalid ID Passed. Use a Valid ID'. Below this, there's a section titled 'List Movies' with filters for 'Title' and 'Year', a 'Sort' dropdown set to 'Title', and a 'Order' dropdown set to 'ASC'. There are also 'Filter' and 'Clear' buttons. The main area is titled 'Latest Movies' and lists movies from IDs 522 to 527. Each row includes columns for 'id', 'title', 'year', and 'Actions' (with 'View', 'Edit', and 'Delete' buttons).

| id  | title                | year | Actions  |
|-----|----------------------|------|--|
| 522 | Charlie: A Toy Story | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 523 | Making Toy Story     | 1985 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 524 | Toy Story 2          | 1999 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 525 | Toy Story 3          | 2010 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 526 | Toy Story 4          | 2018 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 527 | Toy Story of Terror! | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |

| 628 | Toy Story That Time Forgot |  | 2014 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
|-----|----------------------------|--|------|----------------------|----------------------|------------------------|
| 629 | Toy Story                  |  | 1995 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 630 | IT202 The Movie            |  | 2024 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 631 | Captain Boomerang          |  | 1953 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |

This screenshot shows the id being missing and being sent back to the list\_movies page with a message shown to the user.

### Checklist Items (1)

#2 A missing id should redirect back to the list page with an applicable message

### Task #2 - Points: 1

## Text: Screenshots of the details page code

## Checklist

\*The checkboxes are for your own tracking

| #    | Points | Details  |
|------|--------|--|
| ■ #1 | 1      | Show how id is fetched   |
| ■ #2 | 1      | Show the DB query to get the record                                |
| ■ #3 | 1      | Show the code related to presenting the data and showing the links |
| ■ #4 | 1      | Include ucid/date comments for each code screenshot                |
| ■ #5 | 1      | Clearly caption screenshots  |

## Task Screenshots:

### Gallery Style: Large View

**Small              Medium              Large**

```

59         <td>
60             <a href="#" data-per_id="1" data-name="movies.php?id=1" data-broker="1" class="list-item-primary">Edit</a>
61             <a href="#" data-per_id="1" data-name="delete_movies.php?id=1" data-broker="1" class="list-item-deleter">Delete</a>
62         </td>
63     </tr>
64   </tbody>
65 </table>
66 //here we need to go up 1 more directory
67 require_once('../_03__includes/functions.php');
68

```

This screenshot shows the id being fetched, it shows the db query to get the record of the movie from its id. It also shows the how the data is being displayed with the bootstrap card view. This also has my UCID and date at the top.

### Checklist Items (5)

#1 Show how id is fetched

#2 Show the DB query to get the record

#3 Show the code related to presenting the data and showing the links

#4 Include ucid/date comments for each code screenshot

#5 Clearly caption screenshots

#### Task #3 - Points: 1

Text: Explain how the page works

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details   |
|--|--------|---|
| <input checked="" type="checkbox"/> #1 | 1      | Provide the high-level steps for handling the DB lookup and presenting the data |

#### Response:

The movie is fetched from the Get array and if it isn't set the id is set to -1 and will not be found. The empty array named broker is created to then store the movie data. If the movie id is greater than -1 the database is called and a SQL query is set to select the movie from the id in the movies table. If the call is successful the results of the movie will be stored in the broker array, if no result is found a flash message appears telling the user and redirecting them back to list\_movies.php. The movie data is then presented using the bootstrap card layout. It creates User list items and prints the results of those items next to its respective name in the card (title, year, source, imdb\_id, id, created, modified). A button is also created under the card linking to the delete\_movies.php and edit\_movies.php which will be called and append the id of the movie to the page URLs.

#### Task #4 - Points: 1

Text: Add related links

#### Checklist

\*The checkboxes are for your own tracking

| # | Points | Details |
|---|--------|---------|
|   |        |         |

|  |   |   |
|--|---|---|
| <input checked="" type="checkbox"/> #1 | 1 | Include the heroku prod link for this page  |
| <input checked="" type="checkbox"/> #2 | 1 | Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature |

URL #1

[https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/view\\_movies.php?id=522](https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/view_movies.php?id=522)

URL #2

<https://github.com/dunnfall/df39-it202-008/pull/57>

### ● Edit Data Page (2 pts.)

[^COLLAPSE ^](#)

#### Task #1 - Points: 1

Text: Screenshots of the edit page

#### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Show before and after screenshots of data you'll edit                |
| <input checked="" type="checkbox"/> #2 | 1      | Show examples of validation messages                                 |
| <input checked="" type="checkbox"/> #3 | 1      | Show an example of successful edit messages                          |
| <input checked="" type="checkbox"/> #4 | 1      | Design/Style should be considered (i.e., bootstrap, custom css, etc) |
| <input checked="" type="checkbox"/> #5 | 1      | Make sure the heroku dev url is visible in the address bar           |
| <input checked="" type="checkbox"/> #6 | 1      | Clearly caption screenshots  |

Task Screenshots:

#### Gallery Style: Large View

Small      Medium      Large

The screenshot shows a web browser window with the following details:

- Address Bar:** df39-it202-008-dev-1606b59608c7.herokuapp.com
- Open Tabs:**
  - df39-it202-008-dev - GitHub | Heroku
  - https://df39-it202-008-dev-1606b59608c7.. - GitHub
  - df39-it202-008-prod - GitHub | Heroku
  - https://df39-it202-008-prod-tb39da6847aa.. - GitHub
  - Create Singular Mode Data Page by dunnfall - ..
- Header:** Movie Watch Home Profile Create/Fetch Movies List/Edit Movies Logout
- Form Title:** Edit Movie
- Form Fields:**
  - Movie Name: Charlie's Toy Story
  - Movie Year: 2013
  - MDB ID: tt2475692
- Buttons:** Update

Heroku bar shown in URL, before picture of editing the Charlie: A Toy Story Movie. Design and styling done with bootstrap.

#### Checklist Items (4)

#1 Show before and after screenshots of data you'll edit

#4 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#5 Make sure the heroku dev url is visible in the address bar

#6 Clearly caption screenshots



Successful edit message shown

#### Checklist Items (4)

#3 Show an example of successful edit messages

#4 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#5 Make sure the heroku dev url is visible in the address bar

## #6 Clearly caption screenshots

A screenshot of a web browser displaying a movie database application. The URL is <https://cf39-t202-008-dev-169bb59606c7.herokuapp.com>. The page has a yellow header bar with navigation links: Movie Watch, Home, Profile, Create/Fetch Movies, List/Edit Movies, and Logout. Below the header is a search/filter section with fields for Title, Year, Sort (Title), Order (+/-), and Limit (10). Buttons for Filter and Clear are present. The main content area is titled "Latest Movies" and contains a table with the following data:

| id  | title                      | year | Actions  |
|-----|----------------------------|------|--|
| 522 | Charlie A Toy Story        | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 523 | Making Toy Story           | 1995 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 524 | Toy Story 2                | 1999 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 525 | Toy Story 3                | 2010 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 526 | Toy Story 4                | 2019 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 527 | Toy Story of Terror!       | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 528 | Toy Story That Time Forgot | 2014 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 529 | Toy Story                  | 1995 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 530 | IT202 The Movie            | 2024 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 531 | Captain Scarface           | 1963 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 532 | Scarface                   | 1983 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |

Before edit of data on list view page

## Checklist Items (1)

### #1 Show before and after screenshots of data you'll edit

A screenshot of the same movie database application after an edit. The URL is <https://cf39-t202-008-dev-169bb59606c7.herokuapp.com>. The page structure is identical to the first screenshot. The "Latest Movies" table now shows the following updated data:

| id  | title                      | year | Actions  |
|-----|----------------------------|------|--|
| 522 | Matt Toegel The Movie      | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 523 | Making Toy Story           | 1995 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 524 | Toy Story 2                | 1999 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 525 | Toy Story 3                | 2010 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 526 | Toy Story 4                | 2019 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 527 | Toy Story of Terror!       | 2013 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 528 | Toy Story That Time Forgot | 2014 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 529 | Toy Story                  | 1995 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |
| 530 | IT202 The Movie            | 2024 | <a href="#">View</a> <a href="#">Edit</a> <a href="#">Delete</a> |

|     |                  |      |                      |                      |                        |
|-----|------------------|------|----------------------|----------------------|------------------------|
| 631 | Captain Scarface | 1953 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 632 | Scarface         | 1983 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |

## After of title being changed

### Checklist Items (1)

#1 Show before and after screenshots of data you'll edit

#### Task #2 - Points: 1

Text: Screenshots of edit page code

### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Form should have correct data types for each property being requested  |
| <input checked="" type="checkbox"/> #2 | 1      | Form should have correct validation for each field (HTML, JS, and PHP) |
| <input checked="" type="checkbox"/> #3 | 1      | Successful edit should have a user-friendly message                    |
| <input checked="" type="checkbox"/> #4 | 1      | Any errors should have user-friendly messages                          |
| <input checked="" type="checkbox"/> #5 | 1      | Include any other rules like role guards and login checks              |
| <input checked="" type="checkbox"/> #6 | 1      | Include ucid/date comments for each code screenshot                    |
| <input checked="" type="checkbox"/> #7 | 1      | Clearly caption screenshots  |

### Task Screenshots:

#### Gallery Style: Large View

[Small](#)   [Medium](#)   [Large](#)

```

45 > //0F39 4/19/2024
46 $stock = [];
47 if ($id > -1) {
48     //fetch
49     $db = getDB();
50     $query = "SELECT title, year, imbd_id FROM `Movies` WHERE id = :id";
51     try {
52         $stmt = $db->prepare($query);
53         $stmt->execute([':id' => $id]);
54         $r = $stmt->fetch();
55         if ($r) {
56             $stock = $r;
57         }
58     } catch (PDOException $e) {
59         error_log("Error fetching record: " . var_export($e, true));
60         flash("Error fetching record", "danger");
61     }
62 }
63 } else {
64     flash("Invalid id passed", "danger");
65     die(header("Location: " . get_url("admin/list_movies.php")));
66 }
67 $form = [];
68 if ($stock) {
69     $form = [
70         ["type" => "text", "name" => "title", "label" => "Movie Name", "placeholder" => "Movie Name", "value" => "Movie Title", "rules" => ["required" => "required"]],
71         ["type" => "year", "name" => "year", "label" => "Movie Year", "placeholder" => "Movie Year", "value" => "Year of Release", "rules" => ["required" => "required"]],
72         ["type" => "text", "name" => "imbd_id", "label" => "IMDB ID", "placeholder" => "Movie IMDB ID", "value" => "IMDB ID", "rules" => ["required" => "required"]],
73     ];
74     $keys = array_keys($stock);
75
76     foreach ($form as $k => $v) {
77         if (in_array($v["name"], $keys)) {
78             $form[$k]["value"] = $stock[$v["name"]];
79         }
80     }
81 }
82
83 <div class="container-fluid">
84     <h3>Edit Movie</h3>
85 
```

```

87 | <?php
88 |     <?php foreach ($form as $k => $v) { 
89 |         render_input($v);
90 |     } ?>
91 |     <?php render_button(["text" => "Search", "type" => "submit", "text" => "Update"]); ?>
92 | </form>
93 |
94 | </div>
95 |

```

This screenshot shows the correct data type being shown for the form. It also shows a flash message if not able to find the correct if a movie being requested. This does not have any roles or login checks. Has my UCID and date.

## Checklist Items (1)

#1 Form should have correct data types for each property being requested

```

1  <?php
2  //note we need to go up 1 more directory
3  require(__DIR__ . "/../../../../partials/nav.php");
4  ?>
5  <!-- DF39 4/19/2024 -->
6  <?php
7  $id = $_GET["id", -1, false];
8  //1000 handle stock fetch
9  if (isset($_POST["title"])) {
10     foreach ($_POST as $k => $v) {
11         if (!in_array($k, ["title", "year", "imdb_id"])) {
12             unset($_POST[$k]);
13         }
14         $quote = $_POST;
15         error_log("Cleaned up POST: " . var_export($quote, true));
16     }
17     //insert data
18     $db = getDB();
19     $query = "UPDATE `Movies` SET ";
20
21     $params = [];
22     //per record
23     foreach ($quote as $k => $v) {
24
25         if ($params) {
26             $query .= ",";
27         }
28         //be sure $k is trusted as this is a source of sql injection
29         $query .= "$k=:{$k}";
30         $params[":{$k}"] = $v;
31     }
32
33     $query .= " WHERE id = :id";
34     $params[":id"] = $id;
35     error_log("Query: " . $query);
36     error_log("Params: " . var_export($params, true));
37     try {
38         $stmt = $db->prepare($query);
39         $stmt->execute($params);
40         flash("Movie Updated!", "success");
41     } catch (PDOException $e) {
42         error_log("Something broke with the query" . var_export($e, true));
43         flash("Movie Name Already Exists, Enter a New Name", "danger");
44     }
45 }

```

This shows a successful edit flashing a Movie Updated flash message.

## Checklist Items (0)

```

//DF39 4/19/24
$form = [];
if ($stock) {
    $form = [
        ["type" => "text", "name" => "title", "label" => "Movie Name", "placeholder" => "Movie Name", "value" => "Movie Title", "rules" => ["required" => "required"]],
        ["type" => "year", "name" => "year", "label" => "Movie Year", "placeholder" => "Movie Year", "value" => "Year of Release", "rules" => ["required" => "required"]],
        ["type" => "text", "name" => "imdb_id", "label" => "IMDB ID", "placeholder" => "Movie IMDB ID", "value" => "IMDB ID", "rules" => ["required" => "required"]],
    ];
    $keys = array_keys($stock);
}

```

HTML validation at the end of the fields being required.

### Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

#### Task #3 - Points: 1

Text: Screenshot of records from DB

#### Checklist

\*The checkboxes are for your own tracking

| #    | Points | Details  |
|------|--------|--|
| ■ #1 | 1      | Show a before and after screenshot of the record |
| ■ #2 | 1      | Note what differs                                |
| ■ #3 | 1      | Clearly caption screenshots                      |

#### Task Screenshots:

##### Gallery Style: Large View

Small

Medium

Large

The screenshot shows a MySQL Workbench interface with a database named 'df38-h202-C08'. The 'Movies' table is selected, displaying 46 rows. The table structure includes columns for id (primary key), created, modified, title, year, mids\_id, and sources. A search bar at the top contains the text 'df38-h202-C08'. The interface includes standard MySQL Workbench navigation and monitoring tools.

| 1 | 522 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Matt Toogel The Movie | 2013 | tt2475693 |
|---|-----|---------------------|---------------------|-----------------------|------|-----------|
| 2 | 523 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Matt Toogel The Movie | 1965 | tt0245265 |
| 3 | 524 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 2           | 1999 | tt0120363 |
| 4 | 525 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 3           | 2010 | tt0435761 |
| 5 | 526 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 4           | 2019 | tt1079375 |

This screenshot shows Matt Toegel the Movie Record in DB before changing, title, imdb id and year

#### Checklist Items (3)

#1 Show a before and after screenshot of the record

#2 Note what differs

#3 Clearly caption screenshots

The screenshot shows the MySQL Workbench interface with the 'Movies' table selected. The table has columns: id, created, modified, title, year, and imdb\_id. The data is as follows:

|  | id | created             | modified            | title       | year |
|--|----|---------------------|---------------------|-------------|------|
|  | 1  | 2024-04-19 19:48:54 | 2024-04-19 19:48:54 | Toy Story 1 | 1995 |
|  | 2  | 2024-04-19 19:48:54 | 2024-04-19 19:48:54 | Toy Story 2 | 1999 |
|  | 3  | 2024-04-19 19:48:54 | 2024-04-19 19:48:54 | Toy Story 3 | 2010 |
|  | 4  | 2024-04-19 19:48:54 | 2024-04-19 19:48:54 | Toy Story 4 | 2019 |

This after screenshot shows the movie name being changed with the year and imdb id.

#### Checklist Items (3)

#1 Show a before and after screenshot of the record

#2 Note what differs

#3 Clearly caption screenshots

Task #4 - Points: 1

Text: Explain the process

COLLAPSE

### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details  |
|--|--------|--|
| <input checked="" type="checkbox"/> #1 | 1      | Provide a high-level step-by-step of how the fetching of the record, populating the form, and the update works and gets changed in your DB |
| <input checked="" type="checkbox"/> #2 | 1      | Briefly describe the validations for the applicable fields   |

### Response:

The movie id is fetched from the get array and if it isn't set the id is set to -1. If the movie id is greater than -1 the database is called and the SQL query is made to select the movie from the movies table. If an error occurs finding the movie and its data an error is thrown and a flash message is shown to the user. If it is successful the result is stored in the stock array. The array form is made with the structure of the form fields and has keys for type name label placeholder value and rules. In these rules rely the html validation which stops the user from submitting the form if it is empty. The code loops over the fields in the form array and matches a key in the stock array and the value of the field is set to the values from the stock array. After the form is submitted the post array is there to remove any keys that aren't title, year and imdb\_id. The sanitized data is stored in the quote array. Now a sql update query is prepared to insert the updated data in the db. The update query uses the parameters from the quote array and the movie id, if successful the user gets a success flash message. If not successful the user gets a error occurred flash message.

### Task #5 - Points: 1

Text: Add related links

### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details   |
|--|--------|---|
| <input checked="" type="checkbox"/> #1 | 1      | Include the heroku prod link for this page  |
| <input checked="" type="checkbox"/> #2 | 1      | Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature |

### URL #1

<https://github.com/dunnfall/df39-it202-008/pull/54>

### URL #2

[https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/edit\\_movies.php?id=522](https://df39-it202-008-prod-8b39da6847aa.herokuapp.com/Project/admin/edit_movies.php?id=522)

### Delete Handling (1 pt.)

[^COLLAPSE ^](#)

### Task #1 - Points: 1

Text: Screenshots related to delete

### Checklist

\*The checkboxes are for your own tracking

| # | Points | Details |
|---|--------|---------|
|   |        |         |

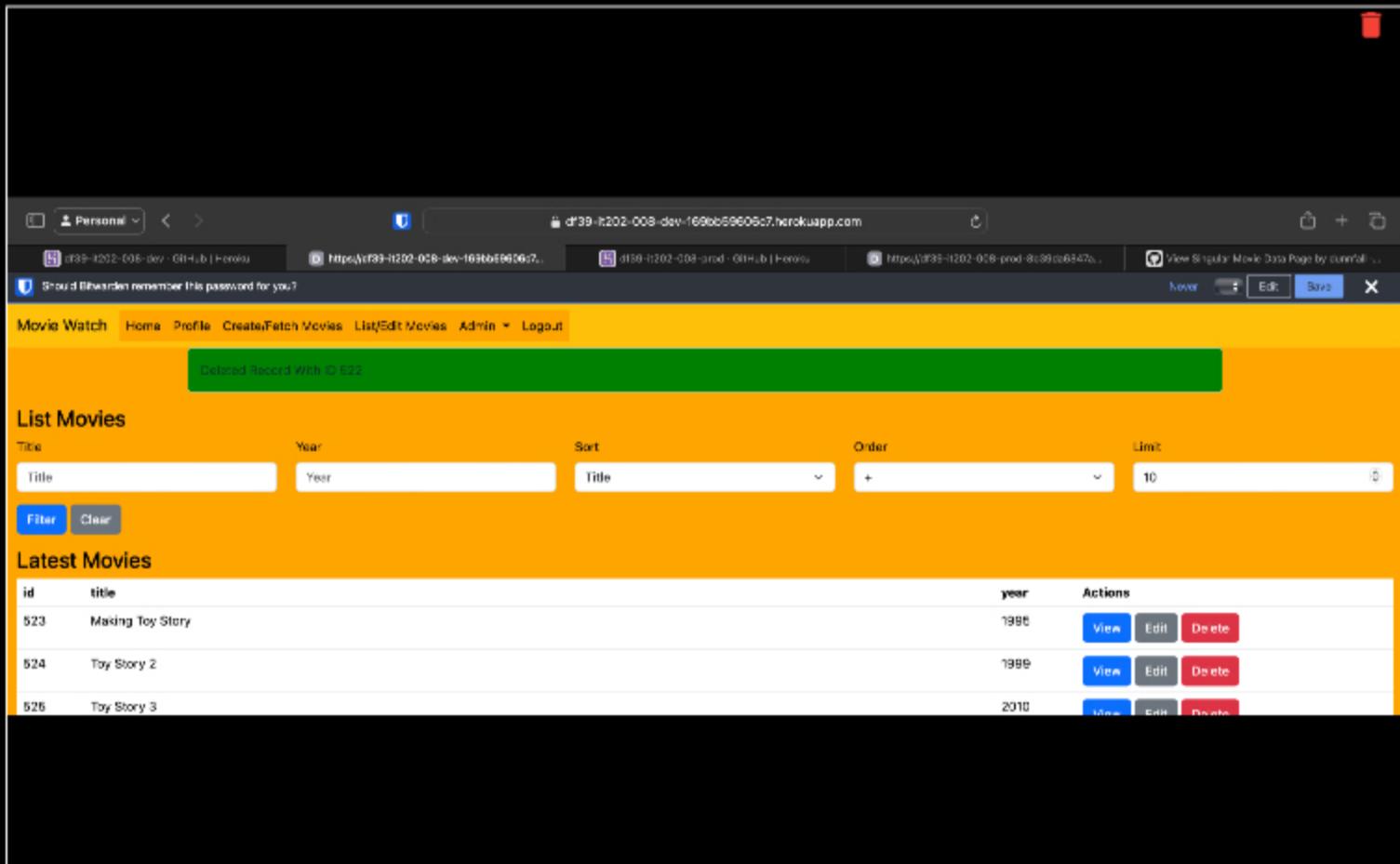
|    |   |   |
|----|---|---|
| #1 | 1 | Show the success message of a delete                                  |
| #2 | 1 | Show any error messages of a failed delete (like id not being passed) |
| #3 | 1 | Show the code related to the delete processing                        |

### Task Screenshots:

**Gallery Style: Large View**

---

Small      Medium      Large



The screenshot shows a web browser window with multiple tabs open. The active tab displays a yellow header with navigation links: Movie Watch, Home, Profile, Create/Edit Movies, List/Edit Movies, Admin, and Logout. Below the header, a green success message box says "Deleted Record With ID 622". Underneath, there are two sections: "List Movies" and "Latest Movies". The "List Movies" section includes search and filter fields for Title and Year, and sorting options for Sort (Title), Order (+/-), and Limit (10). The "Latest Movies" section lists three entries: Toy Story (1995), Toy Story 2 (1999), and Toy Story 3 (2010), each with a "View", "Edit", and "Delete" button. The entire interface has a dark theme with orange and yellow highlights.

Success delete message with id.

### Checklist Items (1)

#1 Show the success message of a delete



The screenshot shows a web browser window with multiple tabs open. The active tab displays a yellow header with navigation links: Movie Watch, Home, Profile, Create/Edit Movies, List/Edit Movies, Admin, and Logout. Below the header, a red error message box says "Invalid ID Passed!". Underneath, there are two sections: "List Movies" and "Latest Movies". The "List Movies" section includes search and filter fields for Title and Year, and sorting options for Sort (Title), Order (+/-), and Limit (10). The "Latest Movies" section lists three entries: Toy Story (1995), Toy Story 2 (1999), and Toy Story 3 (2010), each with a "View", "Edit", and "Delete" button. The entire interface has a dark theme with orange and yellow highlights.

## Latest Movies

| ID  | Title            | Year | Actions              |                      |                        |
|-----|------------------|------|----------------------|----------------------|------------------------|
| 523 | Making Toy Story | 1995 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 524 | Toy Story 2      | 1999 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |
| 525 | Toy Story 3      | 2010 | <a href="#">View</a> | <a href="#">Edit</a> | <a href="#">Delete</a> |

This error message appears when no id is passed.

## Checklist Items (0)

```
1 <?php
2
3 //DF39 4/19/2024
4 session_start();
5 require(__DIR__ . "/../../../../lib/functions.php");
6
7 // require(__DIR__ . "/../../../../lib/nav.php");
8 if (!has_role("Admin")) {
9     flash("You don't have permission to view this page", "warning");
10    die(header("Location: " . get_url("admin/list_movies.php")));
11 }
12
13
14 $id = set($_GET, "id", -1, false);
15 if ($id < 1) {
16     flash("Invalid Id Passed", "danger");
17     die(header("Location: " . get_url("admin/list_movies.php")));
18 }
19
20 $db = getDB();
21 $query = "DELETE FROM `Movies` WHERE id = :id";
22 try {
23     $stmt = $db->prepare($query);
24     $stmt->execute([":id" => $id]);
25     flash("Deleted Record With ID $id", "success");
26 } catch (Exception $e) {
27     error_log("Error Deleting Movie $id" . var_export($e, true));
28     flash("Error Deleting Movie", "danger");
29 }
30 die(header("Location: " . get_url("admin/list_movies.php")));
31
32
```

## Code of deleting movie

## Checklist Items (1)

#3 Show the code related to the delete processing

### Task #2 - Points: 1

Text: Screenshots of the data

## Checklist

\*The checkboxes are for your own tracking

| # | Points | Details   |
|---|--------|---|
| 1 | 1      | Show a before and after screenshot of the DB data |

|  |    |   |   |
|--|----|---|---|
|  | #2 | 1 | Note what changed (i.e., record removed or soft delete value changed) |
|  | #3 | 1 | Clearly caption screenshots   |

Task Screenshots:

### Gallery Style: Large View

[Small](#)    [Medium](#)    [Large](#)

|   | id | created timestamp   | modified timestamp  | title                | year | imdb_id   | source |
|---|----|---------------------|---------------------|----------------------|------|-----------|--------|
| > | 1  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Making Toy Story     | 1995 | tt0245266 | api    |
| > | 2  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 2          | 1999 | tt0120363 | api    |
| > | 3  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 3          | 2010 | tt0435761 | api    |
| > | 4  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 4          | 2019 | tt1979376 | api    |
| > | 5  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story of Terror! | 2013 | tt2446040 | api    |

### Making Toy Story movie shown in db before hard deleting

#### Checklist Items (1)

#1 Show a before and after screenshot of the DB data

|   | id | created timestamp   | modified timestamp  | title                | year | imdb_id   | source |
|---|----|---------------------|---------------------|----------------------|------|-----------|--------|
| > | 1  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 2          | 1999 | tt0120363 | api    |
| > | 2  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 3          | 2010 | tt0435761 | api    |
| > | 3  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story 4          | 2019 | tt1979376 | api    |
| > | 4  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story of Terror! | 2013 | tt2446040 | api    |
| > | 5  | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Making Toy Story     | 1995 | tt0245266 | api    |

|     |     |                     |                     |                            |      |           |        |
|-----|-----|---------------------|---------------------|----------------------------|------|-----------|--------|
| > 5 | 528 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story That Time Forgot | 2014 | tt3473654 | api    |
| > 6 | 529 | 2024-04-19 19:45:54 | 2024-04-19 19:45:54 | Toy Story                  | 1995 | tt0114709 | api    |
| > 7 | 530 | 2024-04-19 19:48:57 | 2024-04-19 19:48:57 | IT202 The Movie            | 2024 | tt1234567 | manual |

After screenshot of the data being hard deleted from db

### Checklist Items (3)

#1 Show a before and after screenshot of the DB data

#2 Note what changed (i.e., record removed or soft delete value changed)

#3 Clearly caption screenshots

#### Task #3 - Points: 1

Text: Explain the delete logic

### Checklist

\*The checkboxes are for your own tracking

| #                                      | Points | Details   |
|--|--------|---|
| <input checked="" type="checkbox"/> #1 | 1      | Is it a soft or hard delete   |
| <input checked="" type="checkbox"/> #2 | 1      | Are there any necessary roles or restrictions? (can only delete their data, can only be done by admin, etc) |
| <input checked="" type="checkbox"/> #3 | 1      | Provide the high-level steps for handling the DB lookup and handling the delete                             |

### Response:

The deletion from the db is a hard delete and this action can only be performed by admins. The session gets started with the session start function. Then the admin role check occurs and if the user is not an admin they get sent back list\_movies.php. The movie is fetched using the get array using the se function. If the id isn't set from this get array it will be set to -1. If the movie id is -1 an error message will be flashed to the user saying error has occurred and they get redirected to the list\_movies.php page. If the id is greater than one a connection is made to the movies db. A sql delete query is prepared to delete the movie with the given id from the get array. The delete query is executed with the movie id as a parameter, if the query is successful the user gets a flash message saying so, if not successful the user gets a message saying invalid id passed. After the delete finished the user is sent back to list\_movies.php

#### Task #4 - Points: 1

Text: Add the pull request link for the branch related to this feature

### Details:

URL #1

<https://github.com/dunnfall/df39-it202-008/pull/55>

Misc (1 pt.)

[COLLAPSE](#)

### Task #1 - Points: 1

Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

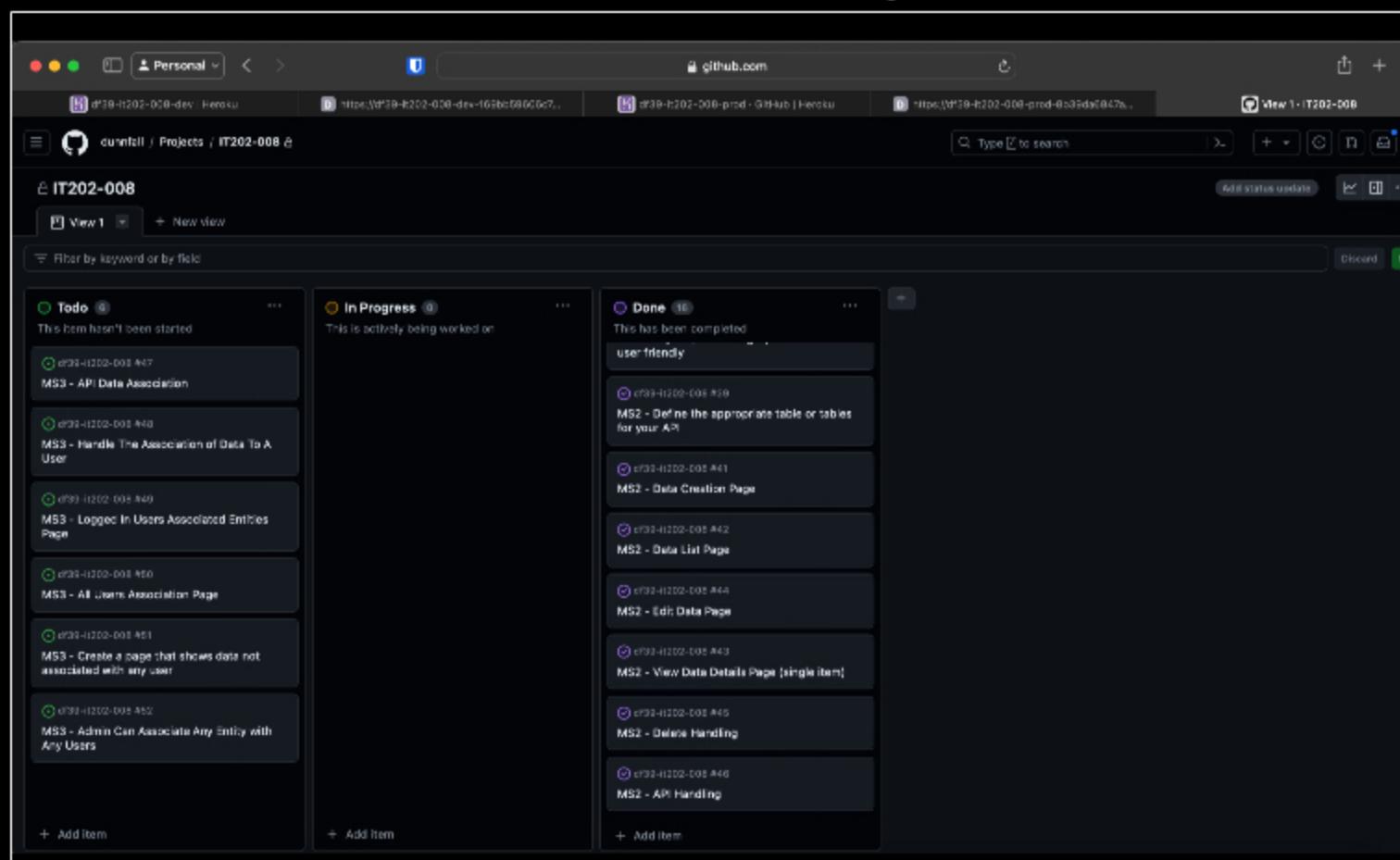
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Screenshot of Project Board on Github

### Task #2 - Points: 1

Text: Provide a direct link to the project board on GitHub

URL #1

### Task #3 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I have a lot of issues at first with my API not working and that was because of the endpoint I was using from my API so I changed to an endpoint where you can query from the title of the movie instead of an ID. I also had some issues with filtering and sorting but resolved it. The issue was simple I just made a mistake about passing the \$params variable into the call for the database to fetch those movies.

### Task #4 - Points: 1

Text: WakaTime Screenshot

#### Details:

Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

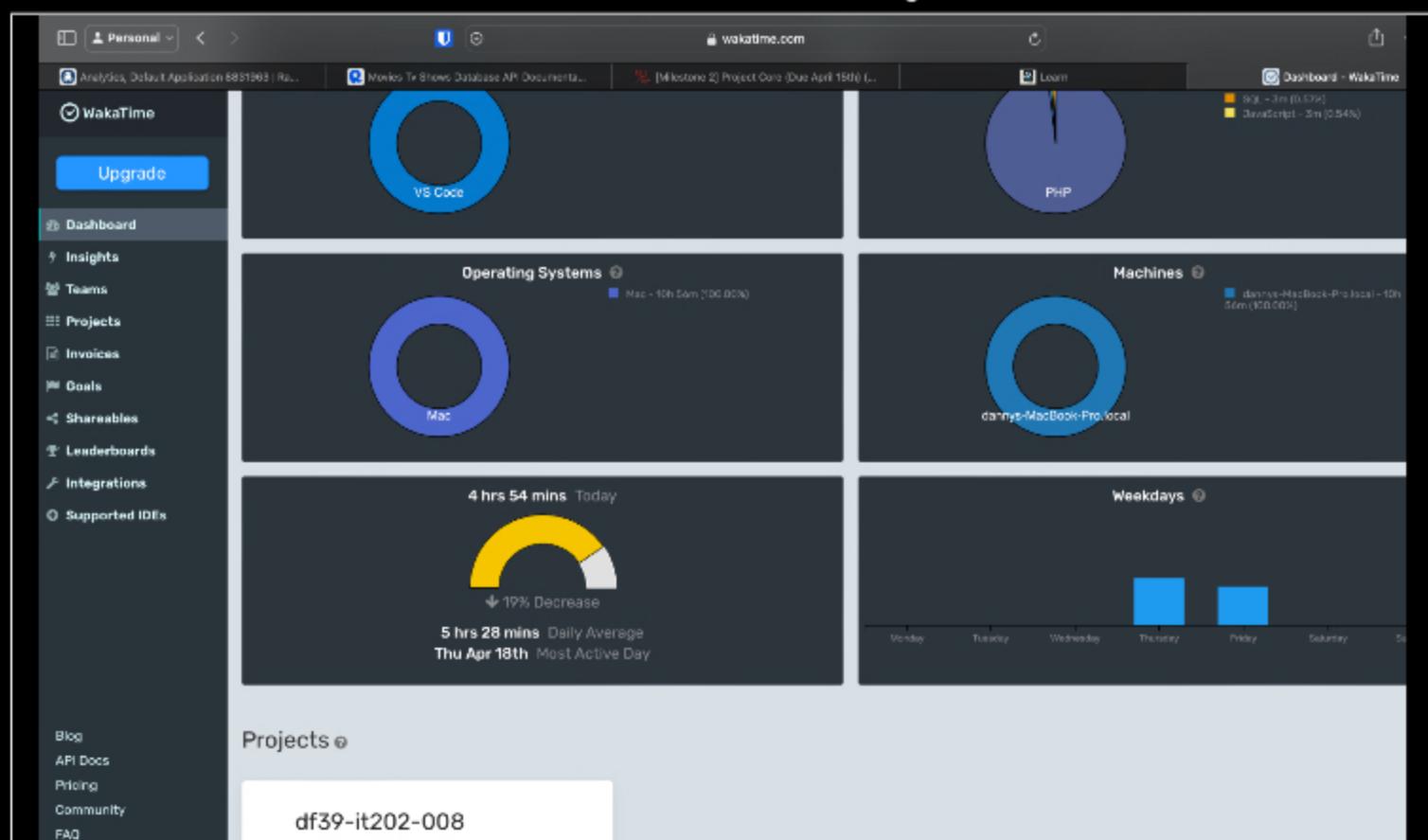
Task Screenshots:

Gallery Style: Large View

Small

Medium

Large



Plugin Status  
About

10 hrs 56 mins

### Wake time screenshot

End of Assignment