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

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 > ...

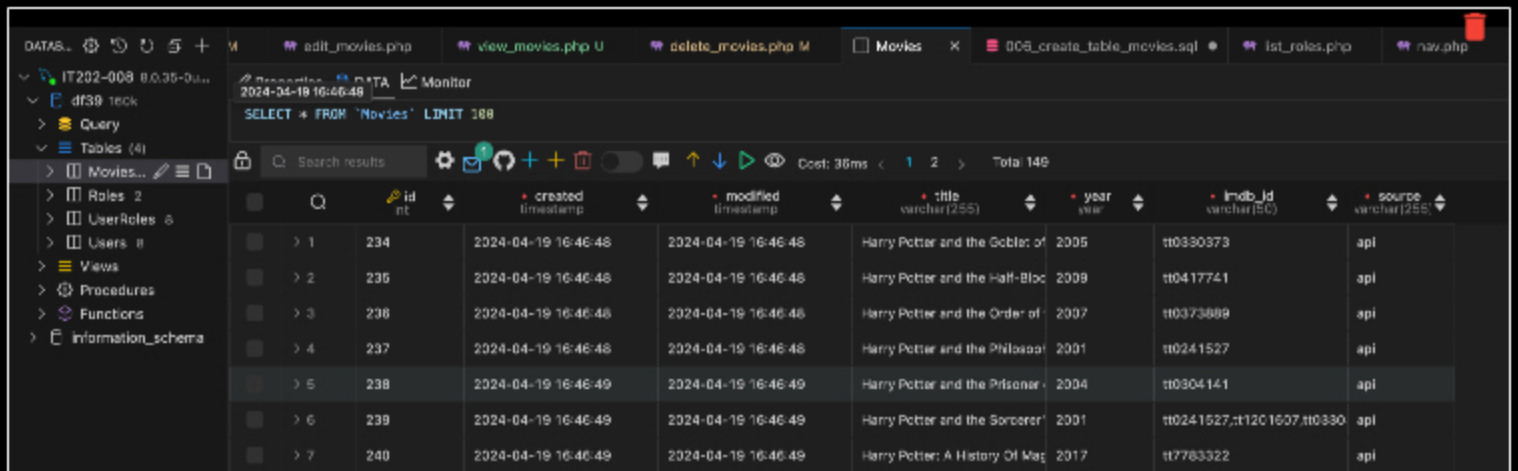
Run | New Tab | Copy | Active Connection

```sql
CREATE TABLE IF NOT EXISTS `Movies` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `created` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    `modified` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    `title` VARCHAR(255) NOT NULL,
    `year` YEAR NOT NULL,
    `imdb_id` VARCHAR(50) NOT NULL,
    `source` VARCHAR(255) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE (`title`)
    -- DF39 4/19/2024
)
```

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



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	tt9609726	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	1989	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	Amphitryon: Happiness from t	1935	tt0026069	api
18	251	2024-04-19 16:46:56	2024-04-19 16:46:56	And the Pursuit of Happiness	1986	tt0092561	api
19	252	2024-04-19 16:46:56	2024-04-19 16:46:56	Arnold Cautious 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

COLLAPSE

Task #2 - Points: 1

Text: Explain the design

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Note the different fields and their purpose
#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.

COLLAPSE

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.

URL #1

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

Data Creation Page (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the creation page

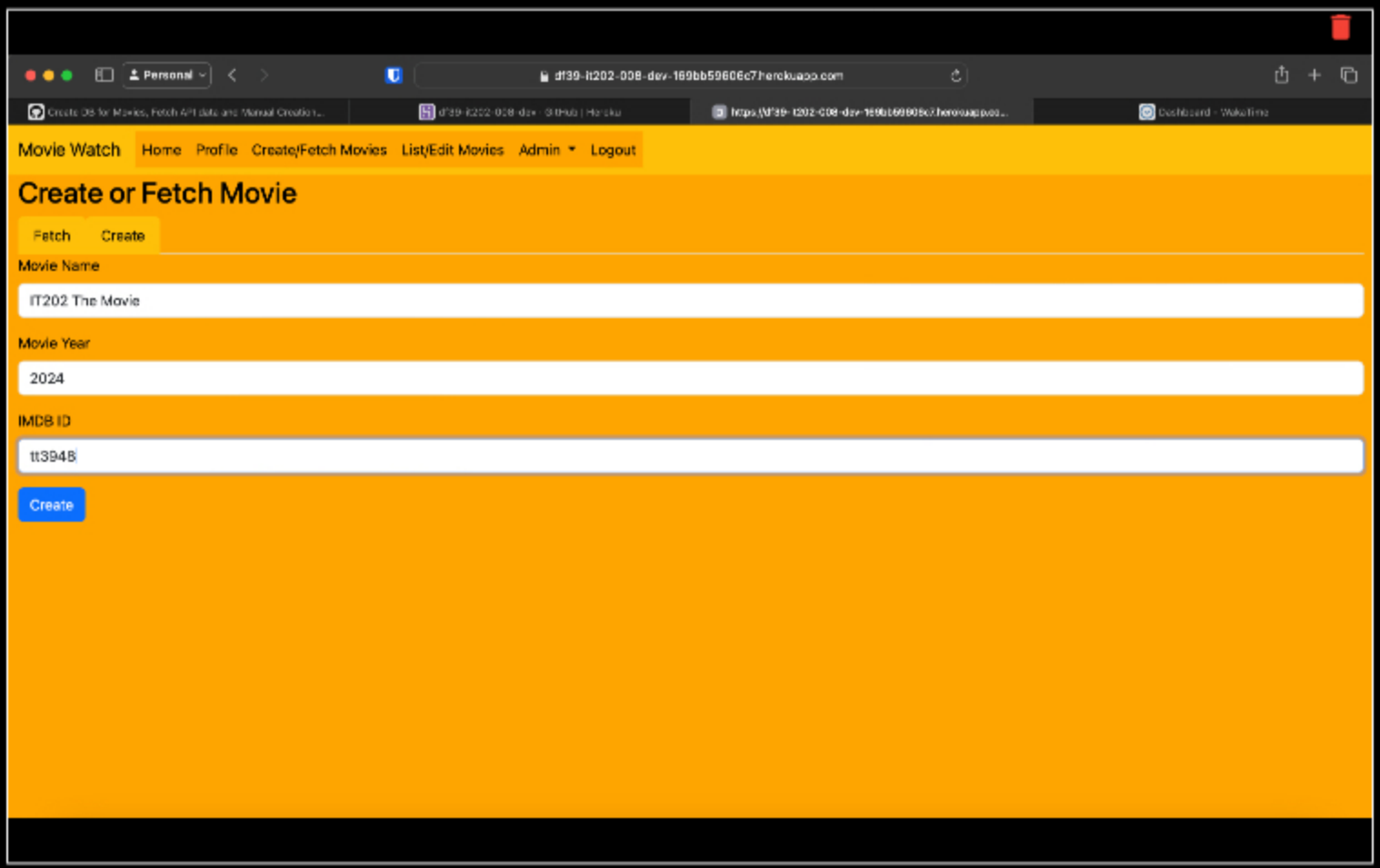
Checklist *The checkboxes are for your own tracking

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

Task Screenshots:

Gallery Style: Large View

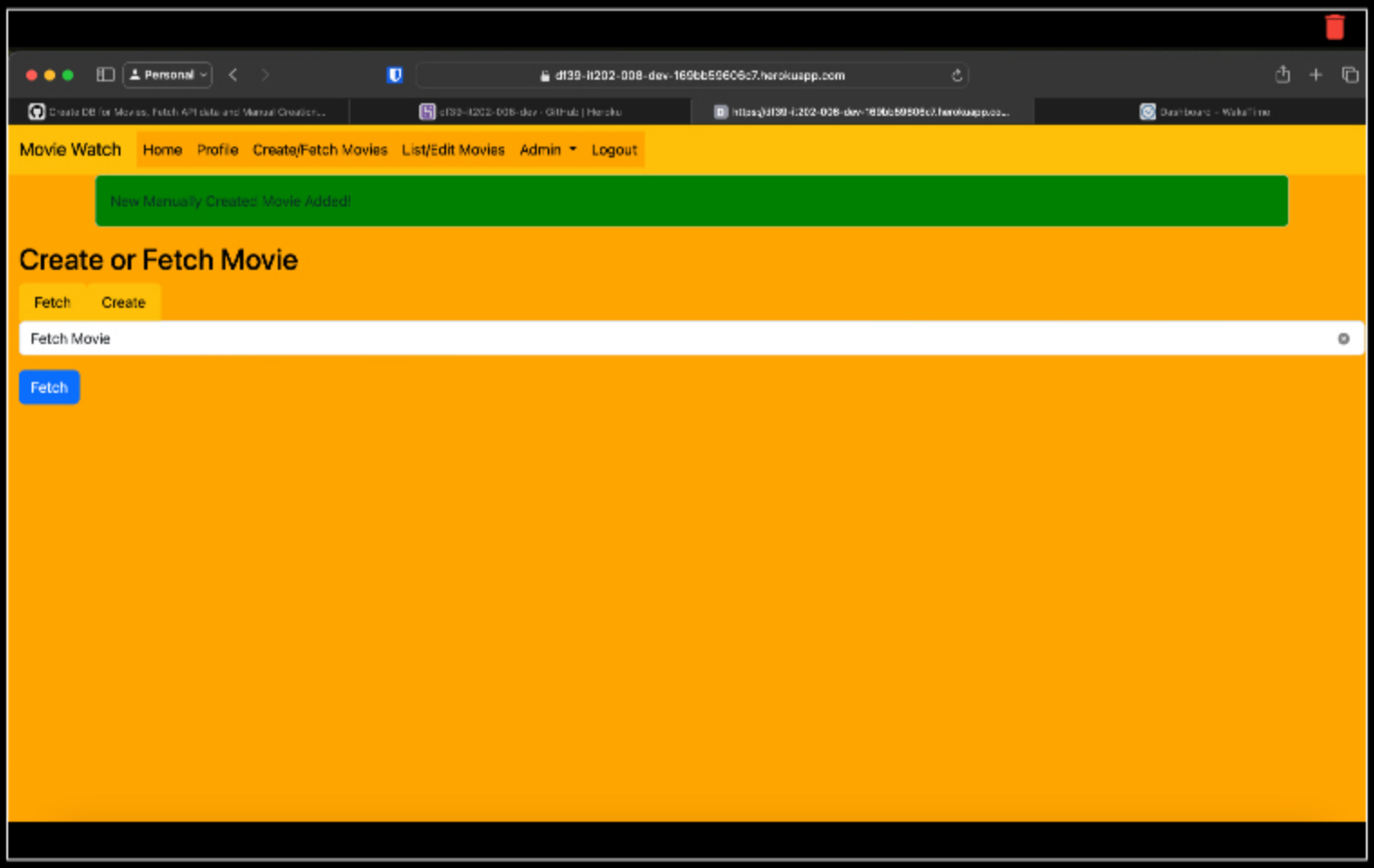
Small Medium Large



This it the custom Creation Page with data being inserted

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

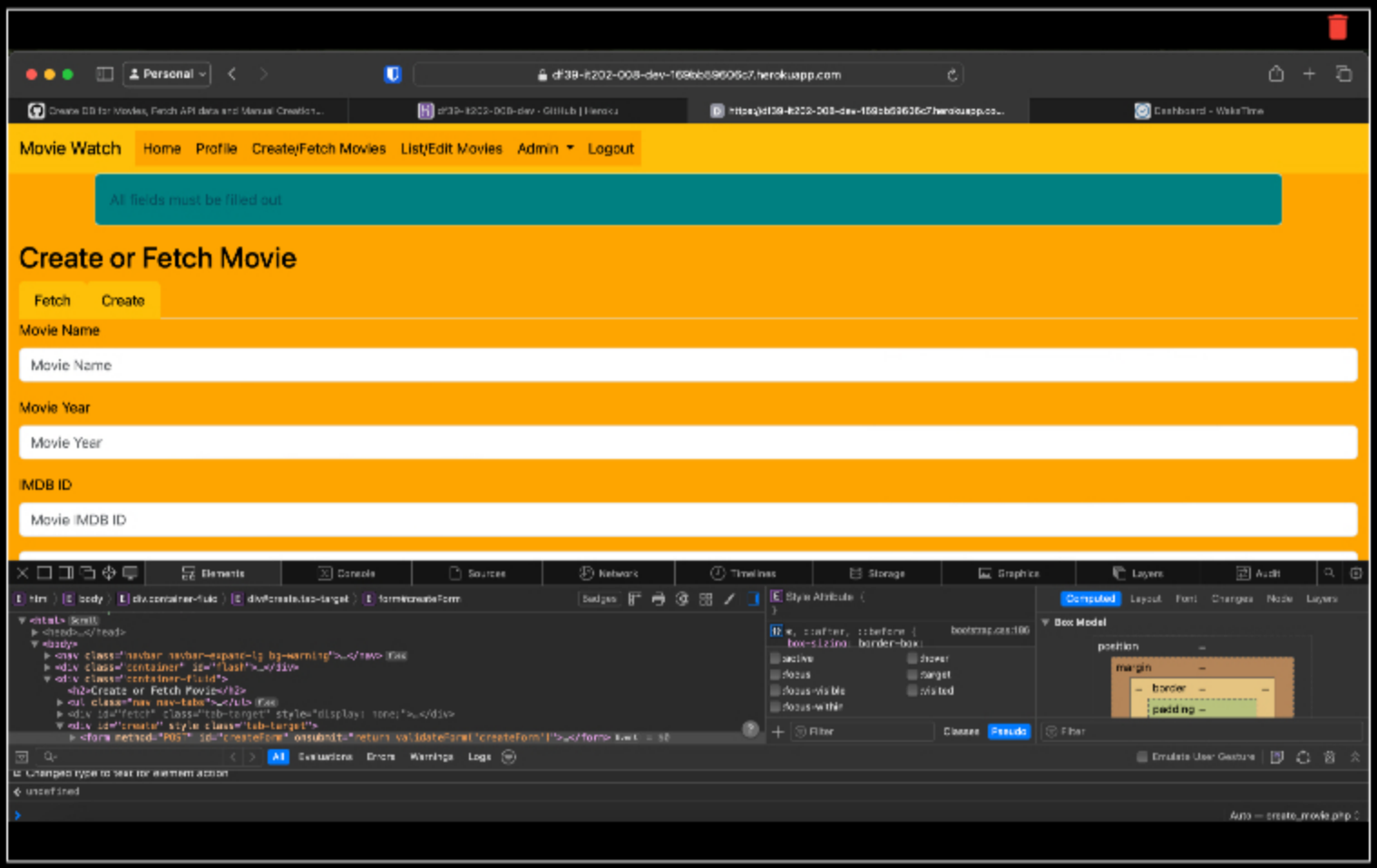
```php
<?php
    //TODO handle movie fetch and movie creation
    //DF39 4/18/2024
    $db = getDB();
    $result = [];

    $endpoint = "https://movies-tv-shows-database.p.rapidapi.com";
    $isRapidAPI = true;
    $rapidAPIHost = "movies-tv-shows-database.p.rapidapi.com";
    $rapidType = "get-movies-by-title";
    $movie_title = "";
    $data = ["title" => "$movie_title"];
    // Fetching Data From API
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        if ($_POST['action'] == 'fetch')
        {
            $movie_title = $_POST["movie_title"];
            $data = ["title" => "$movie_title"];
            $result = get($endpoint, "MOVIE_KEY", $data, $isRapidAPI, $rapidAPIHost);
```

```php
    }
    // 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)

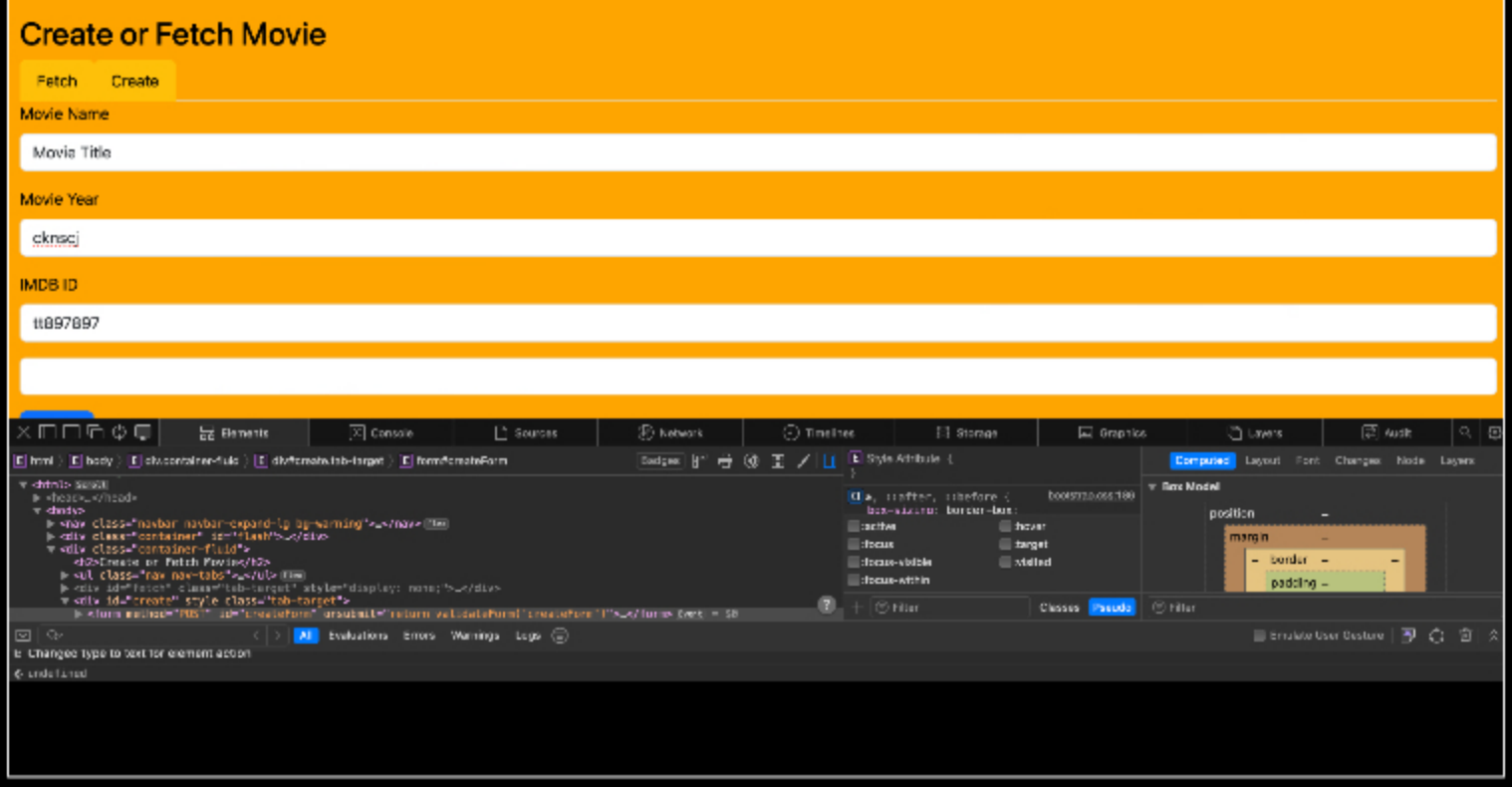
#2 Show how the API data is fetched for API data (must be server-side)



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

Checklist Items (1)

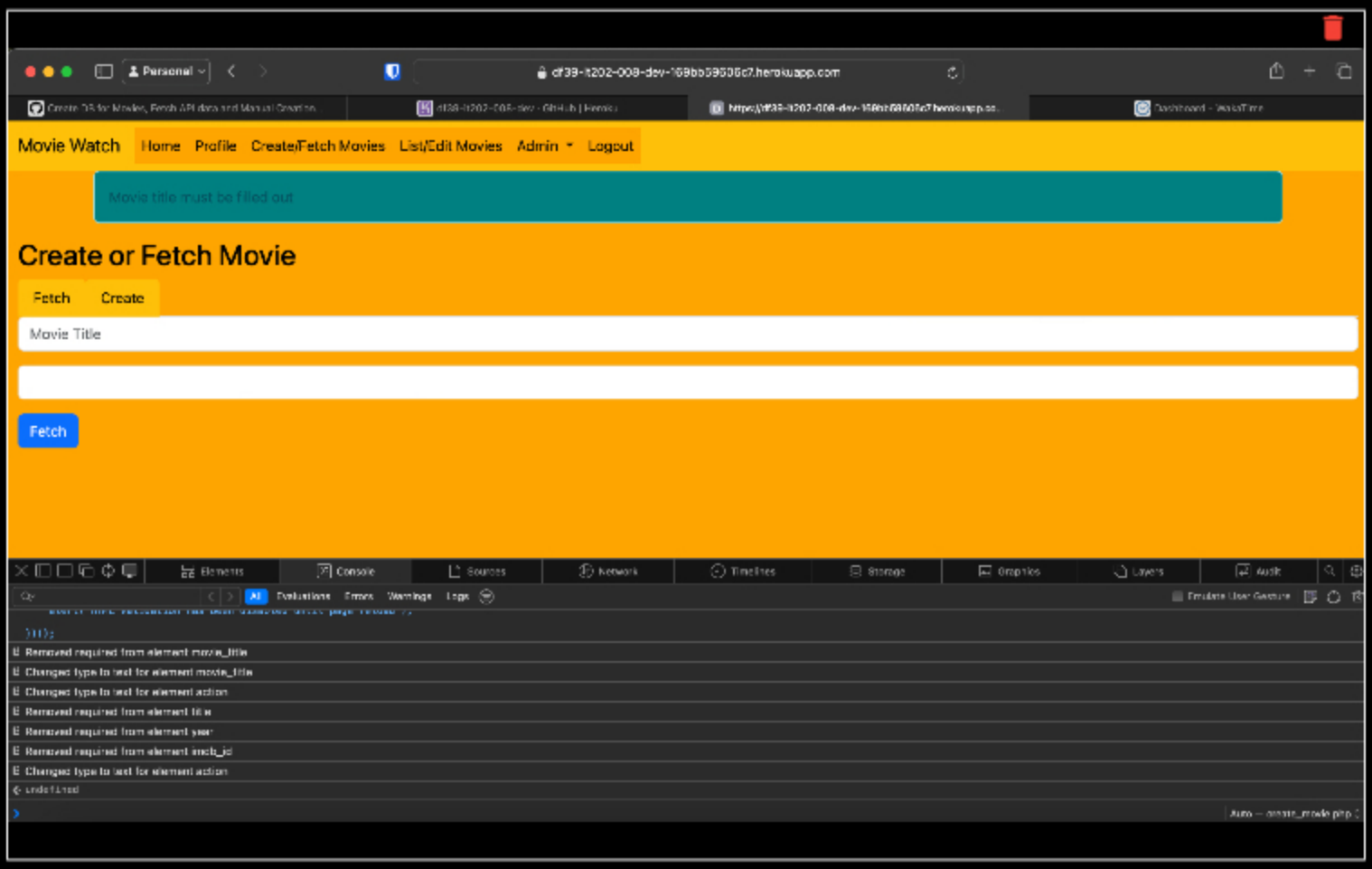
#3 Show examples of validation messages



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



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

Checklist Items (1)

Checklist Items (1)

#3 Show examples of validation messages

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of creation page code

Checklist

*The checkboxes are for your own tracking

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

Task Screenshots:

Gallery Style: Large View

Small Medium Large

public_html > Project > admin > create_movie.php > ...

```php
<?php
//note we need to go up 1 more directory
//DF99 4/15/2024
require(__DIR__ . "/../../partials/nav.php");

?>


<?php if (isset($results)) : ?>
    <?php foreach ($results as $stack) : ?>
        <pre>
            <?php var_export($stack);?>
        </pre>
        <?php endforeach; ?>
        <?php endif; ?>

        <div class="container-fluid">
            <h2>Create or Fetch Movie</h2>
            <ul class="nav nav-tabs">
                <li class="nav-item">
                    <a class="nav-link bg-warning" href="#" onClick="switchTab('create')">Fetch</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link bg-warning" href="#" onClick="switchTab('fetch')">Create</a>
                </li>
            </ul>
            <div id="fetch" class="tab-target">
                <form method="POST" id="fetchForm" onsubmit="return validateForm('fetchForm')">
                    <?php render_input(["type" => "search", "name" => "movie_title", "placeholder" => "Movie Title", "value" => "Fetch Movie", "rules" => ["required" => "required"]]);?>
                    <?php render_input(["type" => "hidden", "name" => "action", "value" => "fetch"]) ?>
                    <?php render_button(["text" => "Fetch", "type" => "submit"]); ?>
                </form>
            </div>
            <div id="create" style="display: none;" class="tab-target">
                <form method="POST" id="createForm" onsubmit="return validateForm('createForm')">
                    <?php render_input(["type" => "text", "name" => "title", "label" => "Movie Name", "placeholder" => "Movie Name", "value" => "Movie Title", "rules" => ["required" => "required"]]);?>
                    <?php render_input(["type" => "year", "name" => "year", "label" => "Movie Year", "placeholder" => "Movie Year", "value" => "Year of Release", "rules" => ["required" => "required"]]);?>
                    <?php render_input(["type" => "text", "name" => "imdb_id", "label" => "IMDB ID","placeholder" => "Movie IMDB ID", "value" => "IMDB ID", "rules" => ["required" => "required"]]);?>

                    <?php render_input(["type" => "hidden", "name" => "action", "value" => "create"]);?>
                    <?php render_button(["text" => "Create", "type" => "submit", "text" => "Create"]); ?>
                </form>
            </div>
            <div class="row ">
                </div>
            </div>
            <script>
                function switchTab(tab)
                {
                    let target = document.getElementById(tab);
                    if (target)
                    {
                        let eles = document.getElementsByClassName("tab-target");
                        for (let ele of eles)
                        {
                            ele.style.display = (ele.id === tab) ? "none" : "";
                        }
                    }
                }
            </script>
```

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 users 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.

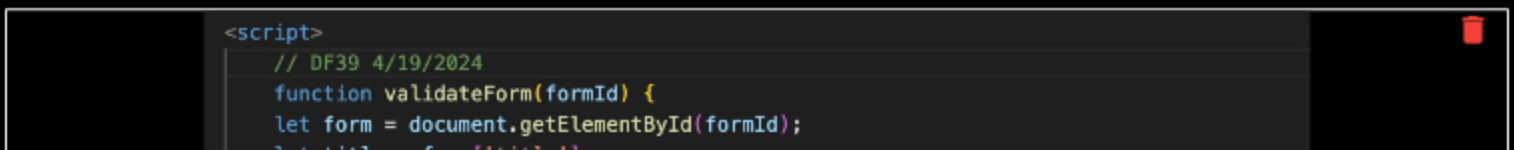
Checklist Items (4)

#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



```
let title = form['title'];
let year = form['year'];
let imdb_id = form['imdb_id'];
let movie_title = form['movie_title'];

if (formId === 'createForm') {
    if (!title.value || !year.value || !imdb_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(imdb_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)

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of records from DB

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show at least one record fetched from the API
#2	1	Show at least one record created via the creation form
#3	1	Note what differs
#4	1	Clearly caption screenshots

Task Screenshots:

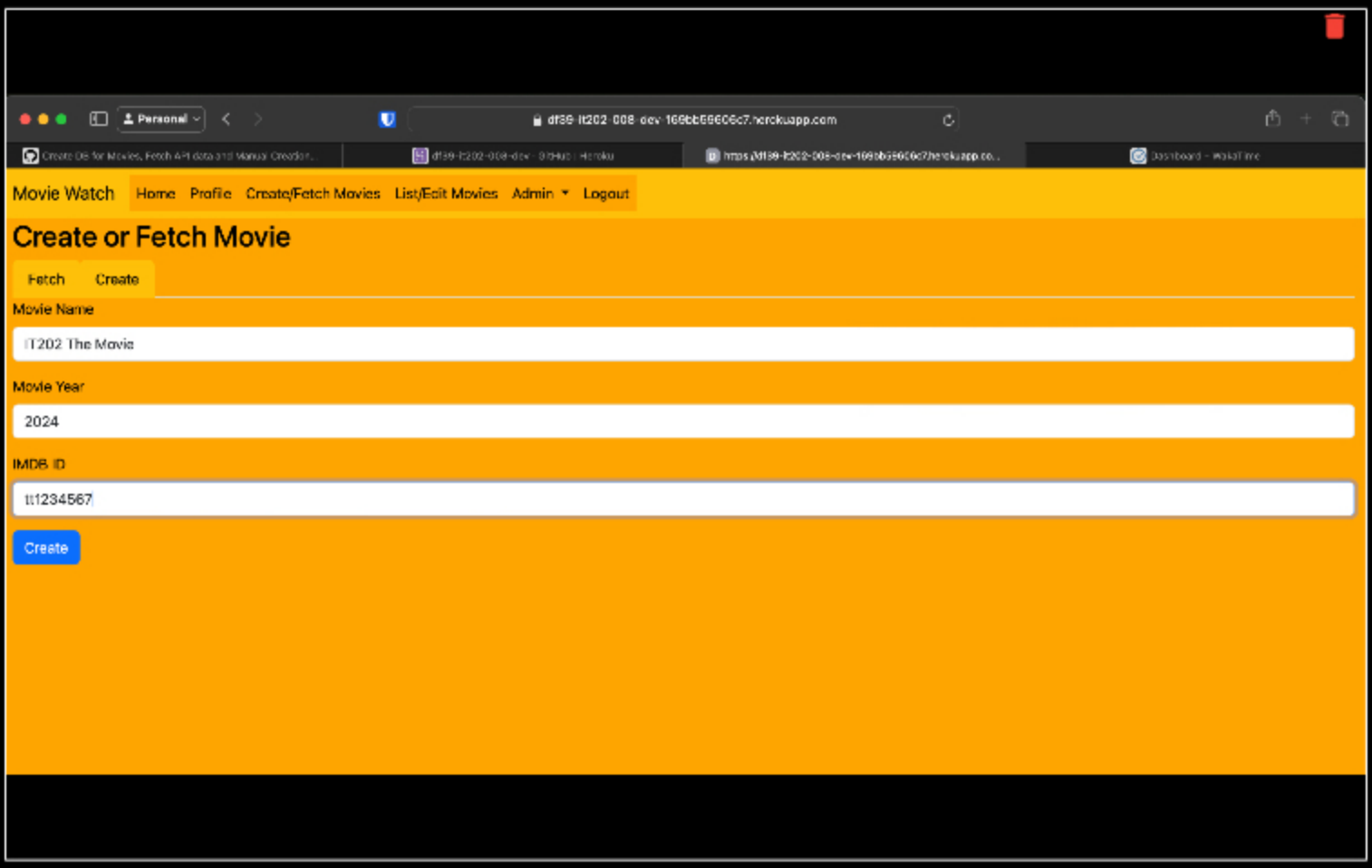
Gallery Style: Large View

Small Medium Large

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

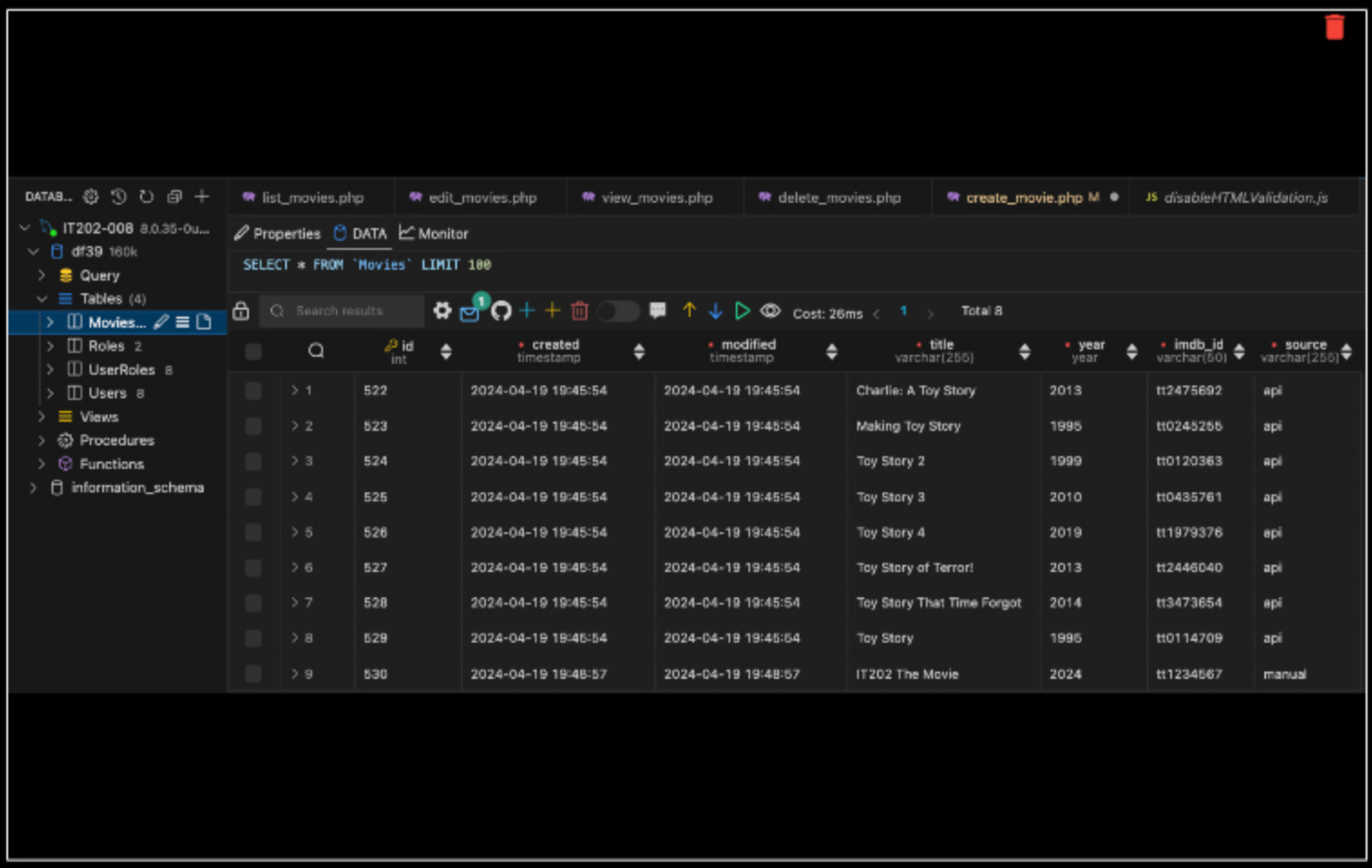


This screen show shows me creating a manual entry for the "IT202 The Movie" movie record to be added into the db.

Checklist Items (2)

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

#4 Clearly caption screenshots



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

COLLAPSE

Task #4 - Points: 1

Text: Explain the process

Checklist *The checkboxes are for your own tracking

#	Points	Details
#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)
#2	1	Provide a high-level step-by-step of how creating custom data works and gets added to

#2	1	your DB (include how duplicates are handled)
#3	1	Briefly describe the validations for the applicable fields
#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 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.

COLLAPSE

Task #5 - Points: 1

Text: Add related links

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#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

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

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the list page

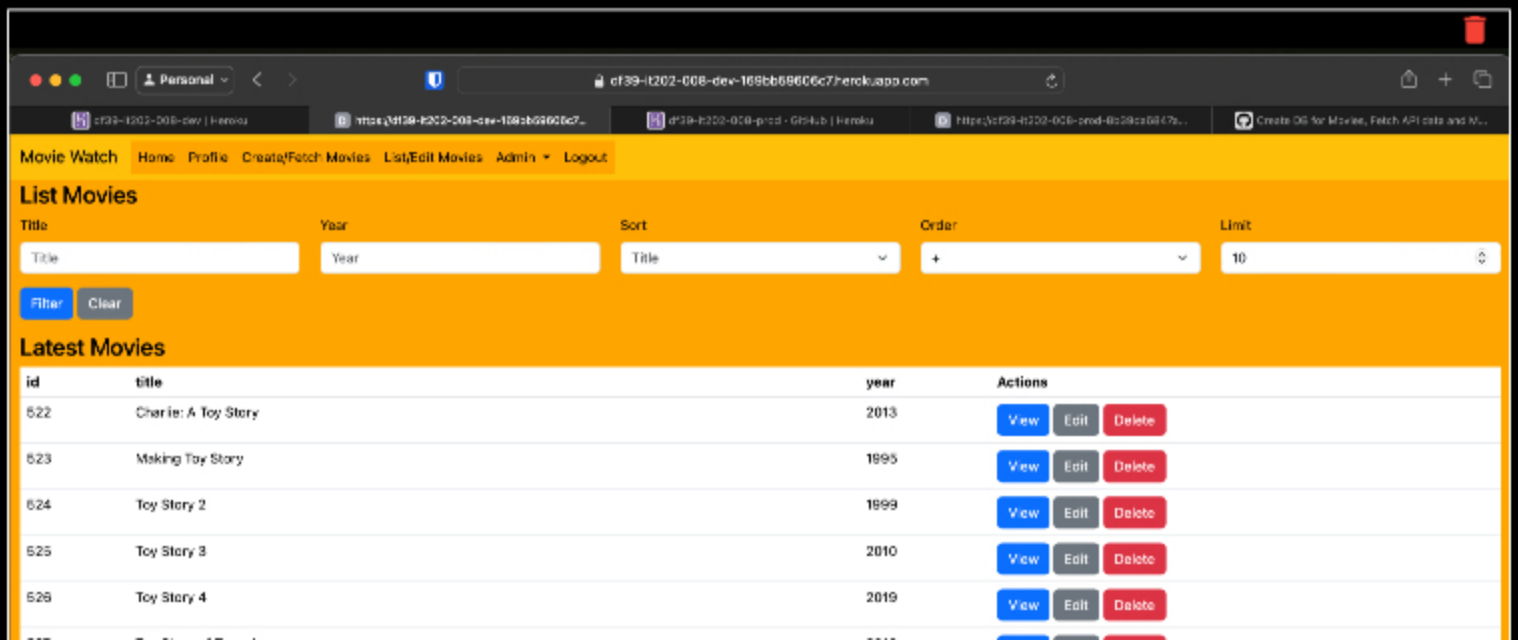
Checklist *The checkboxes are for your own tracking

#	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



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)

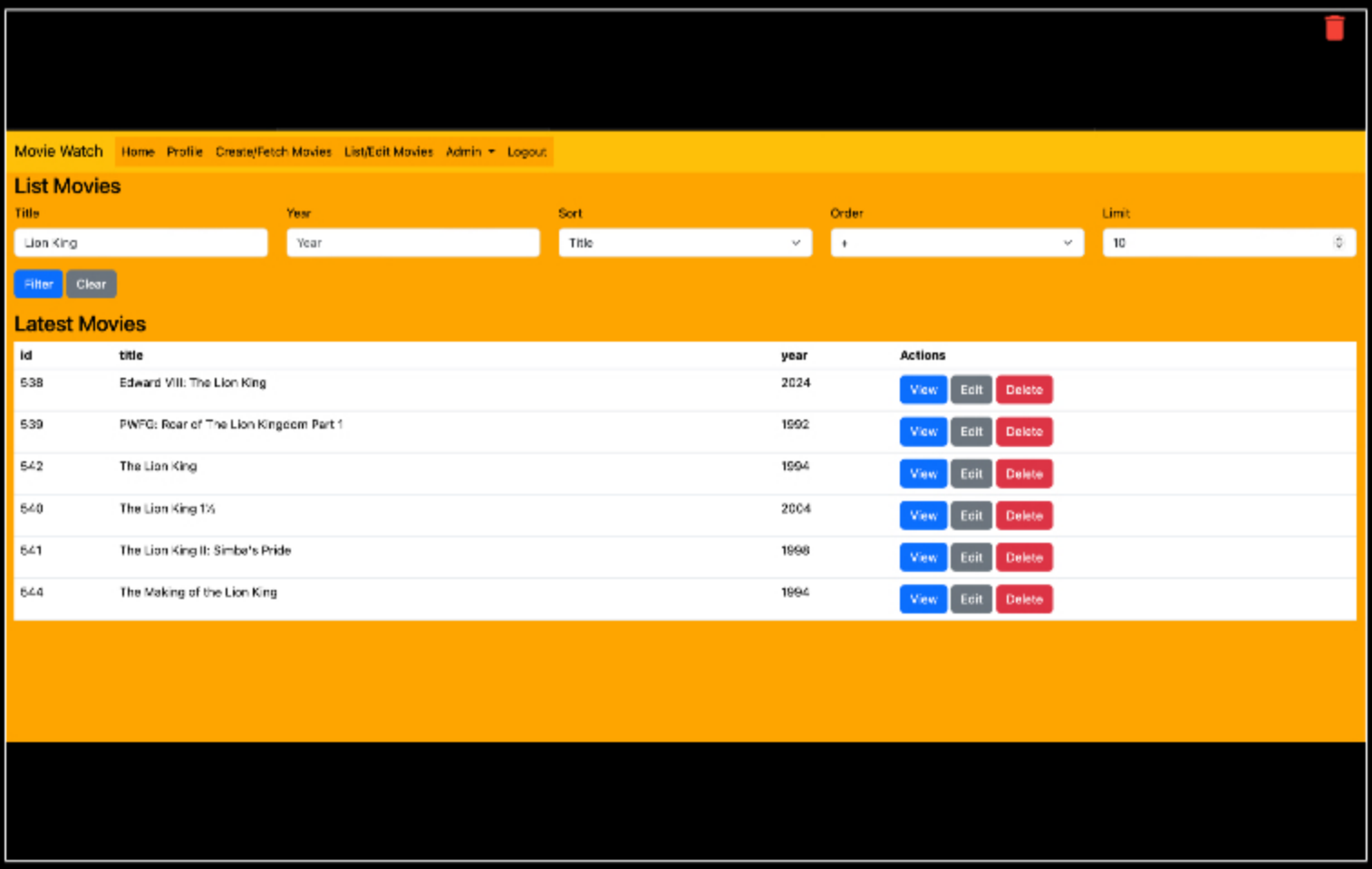
#1 Show the page of your entities listed (have a reasonable number shown)

#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

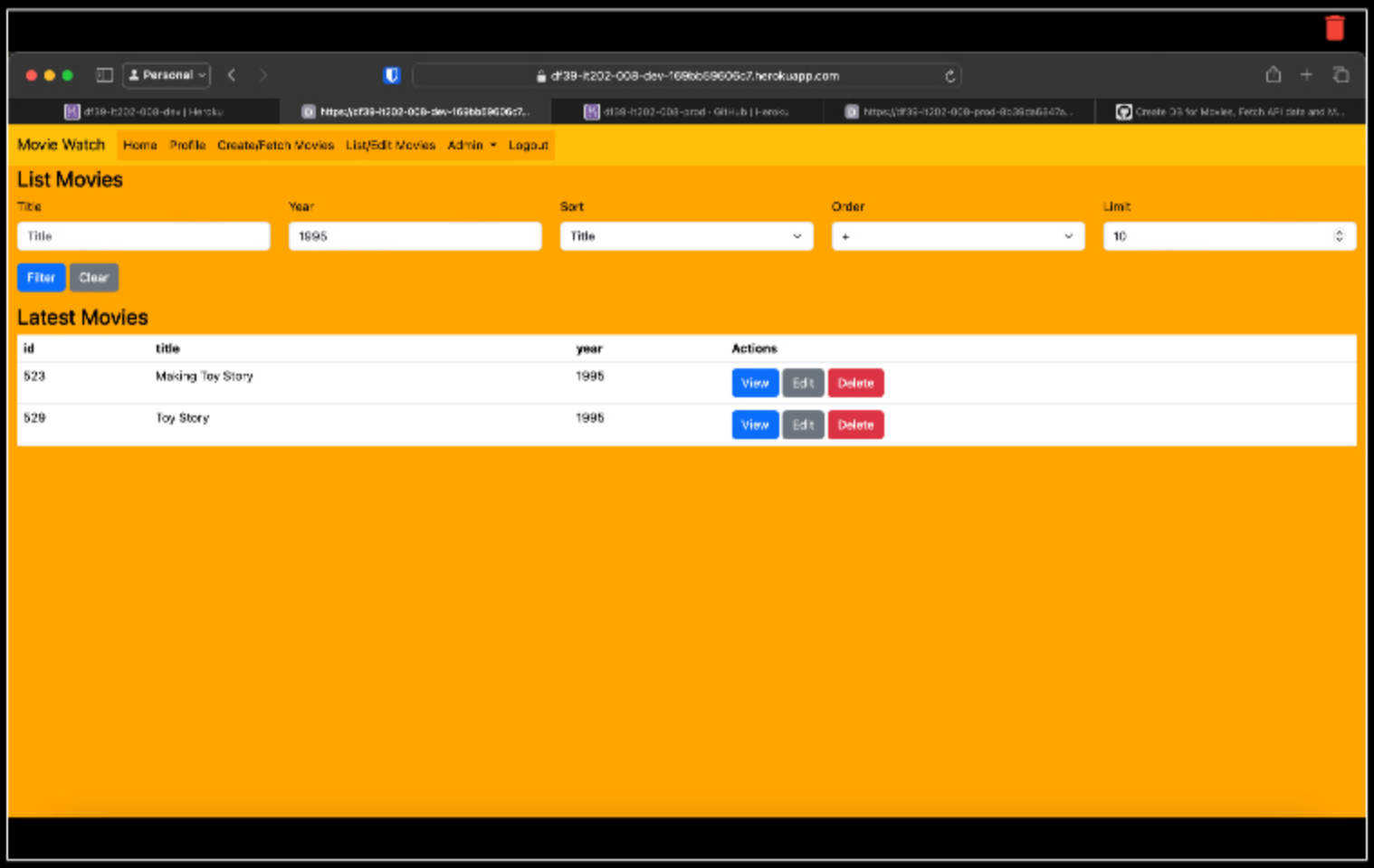


This screenshot shows a sort by title where the title being sorted is lion king and the results are only lion king movies in ascending alphabetical order.

Checklist Items (1)

Checklist Items (1)

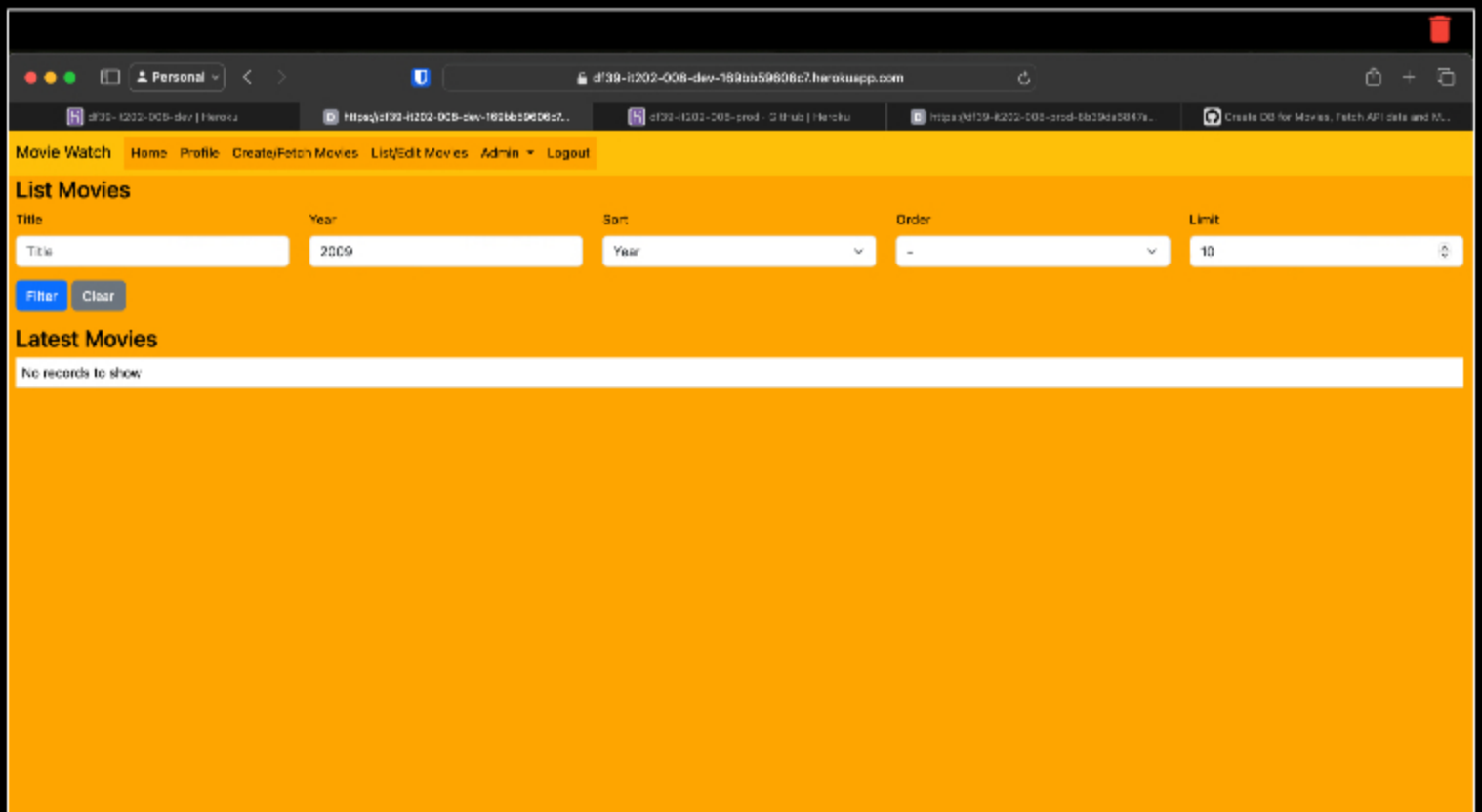
#2 Show the filter/sort form based on your data and the required limit field



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

Checklist Items (1)

#3 Demonstrate a few varied filters/sorts



This screenshot shows a filter of movies by the year of 2009 and not having any records shown with a 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)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the list page code

Checklist

*The checkboxes are for your own tracking

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

Task Screenshots:

Gallery Style: Large View

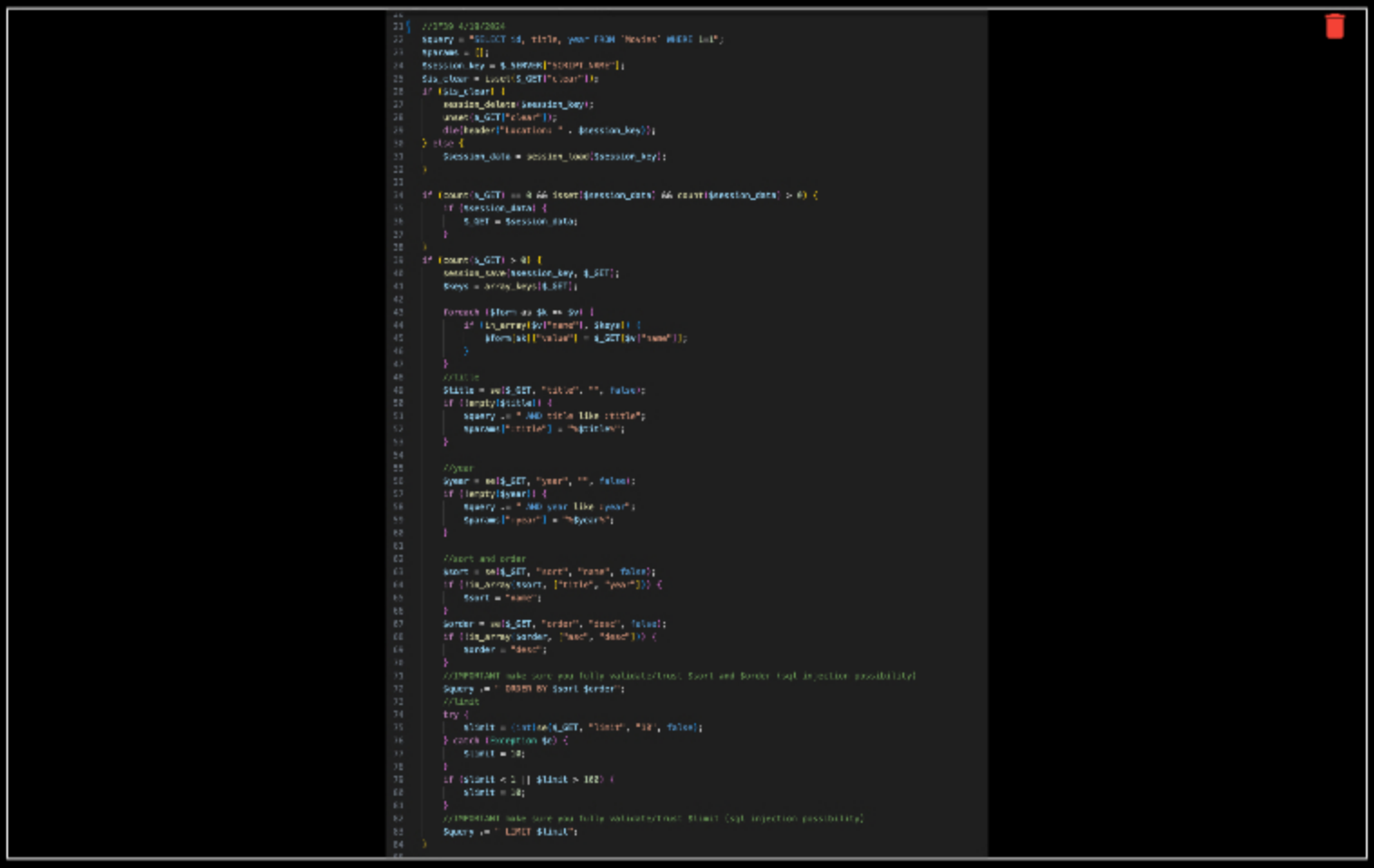
Small Medium Large

```php
<?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)

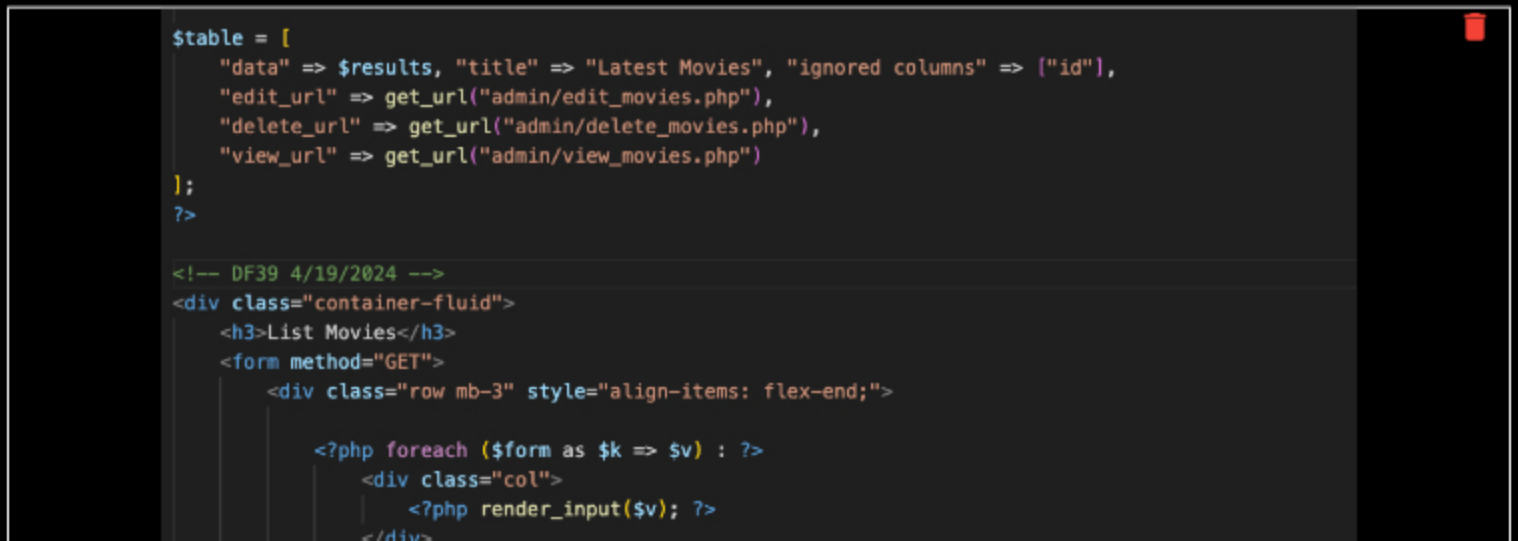
#1 Show the filter/sort form generation



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)



```
            <?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

COLLAPSE

Task #3 - Points: 1

Text: Explain how the page works

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Provide the high-level steps of how the filter/sorting works and how data is displayed
#2	1	Summarize what you're showing on the screen
#3	1	Mention your design/style choice
#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.

COLLAPSE

Task #4 - Points: 1

Text: Add related links

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#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

View Details Page (single entity) (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the details page

Checklist

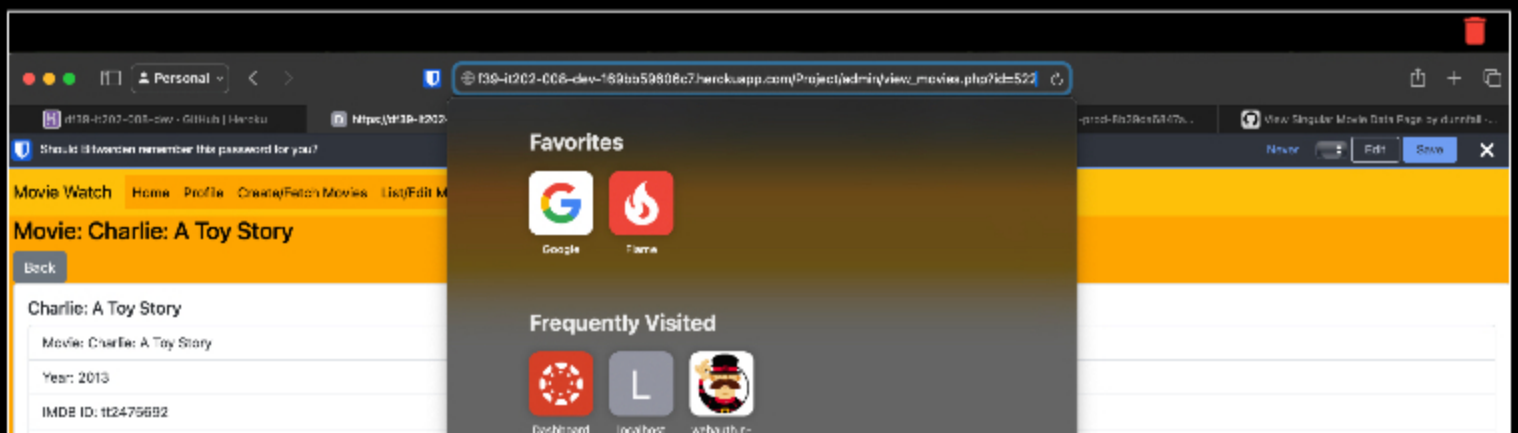
*The checkboxes are for your own tracking

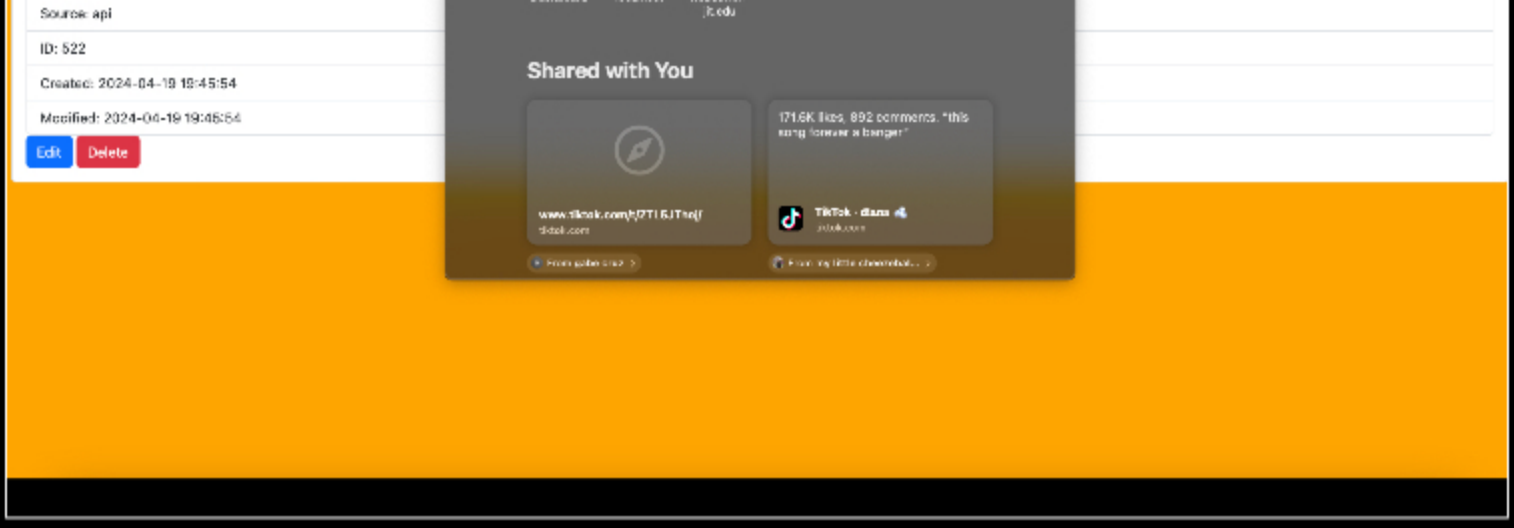
#	Points	Details
#1	1	Entity should be fetch by id (via the url)
#2	1	A missing id should redirect back to the list page with an applicable message
#3	1	Design/Style should be considered (i.e., bootstrap, custom css, etc)
#4	1	Data shown should be more detailed/inclusive than the summary view
#5	1	There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

Task Screenshots:

Gallery Style: Large View

Small Medium Large





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)

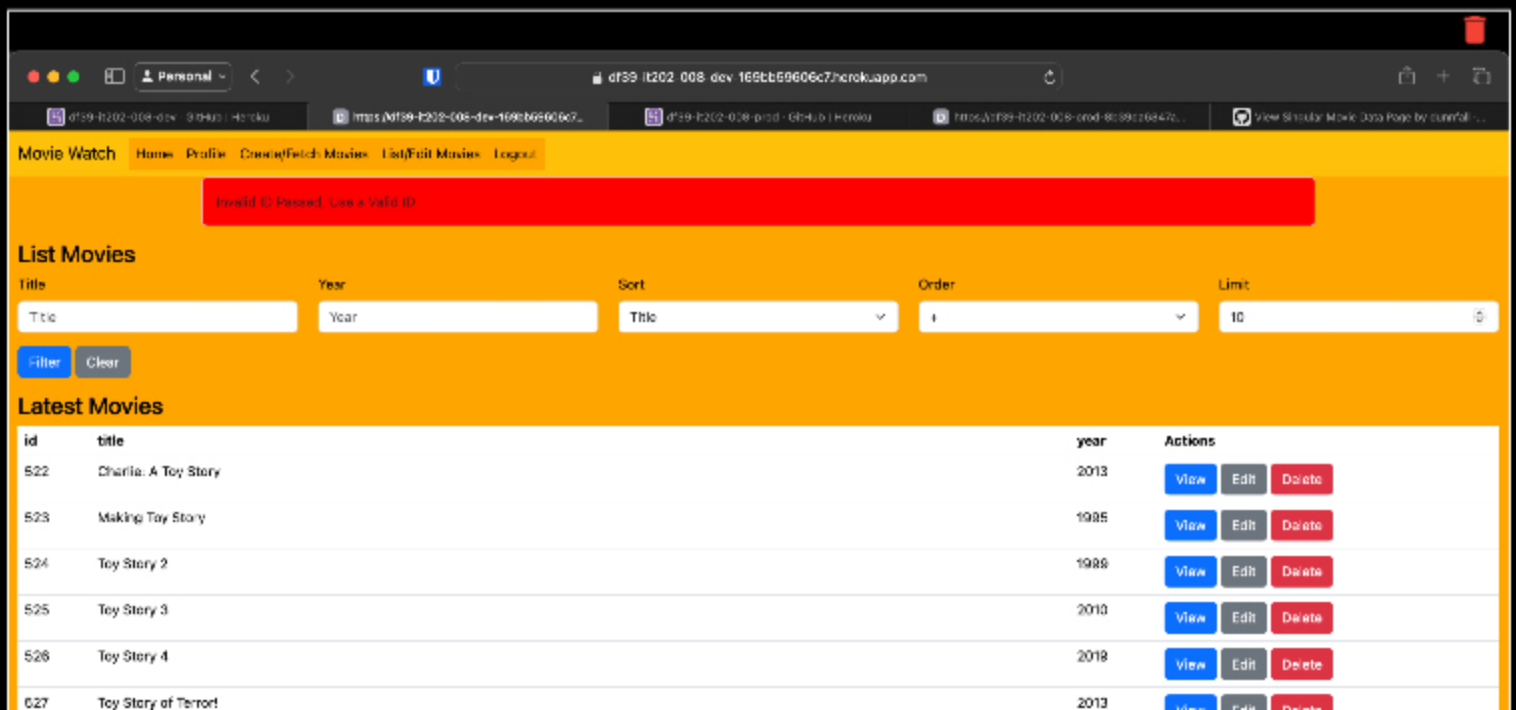
#4 Data shown should be more detailed/inclusive than the summary view

#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 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

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the details page code

Checklist

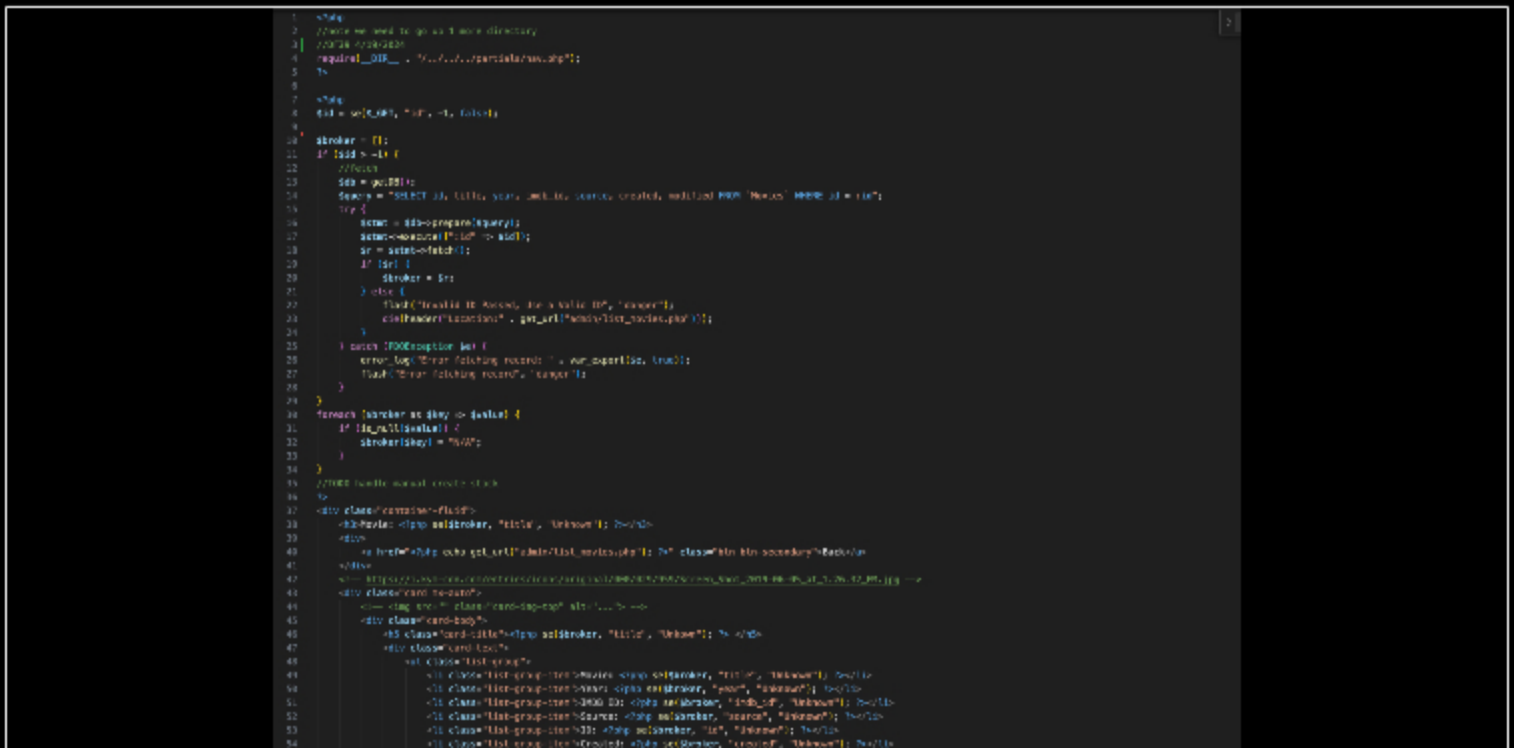
*The checkboxes are for your own tracking

#	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



This screenshot shows the id being fetched, it shows the db query to get the record of the movie form 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

COLLAPSE

Task #3 - Points: 1

Text: Explain how the page works

Checklist *The checkboxes are for your own tracking

#	Points	Details
#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 create to then store the movie data. If the movie id is greater than -1 the data base 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 not 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.

COLLAPSE

Task #4 - Points: 1

Text: Add related links

Checklist *The checkboxes are for your own tracking

#	Points	Details

#1	1	Include the heroku prod link for this page
#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

URL #2

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

Edit Data Page (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the edit page

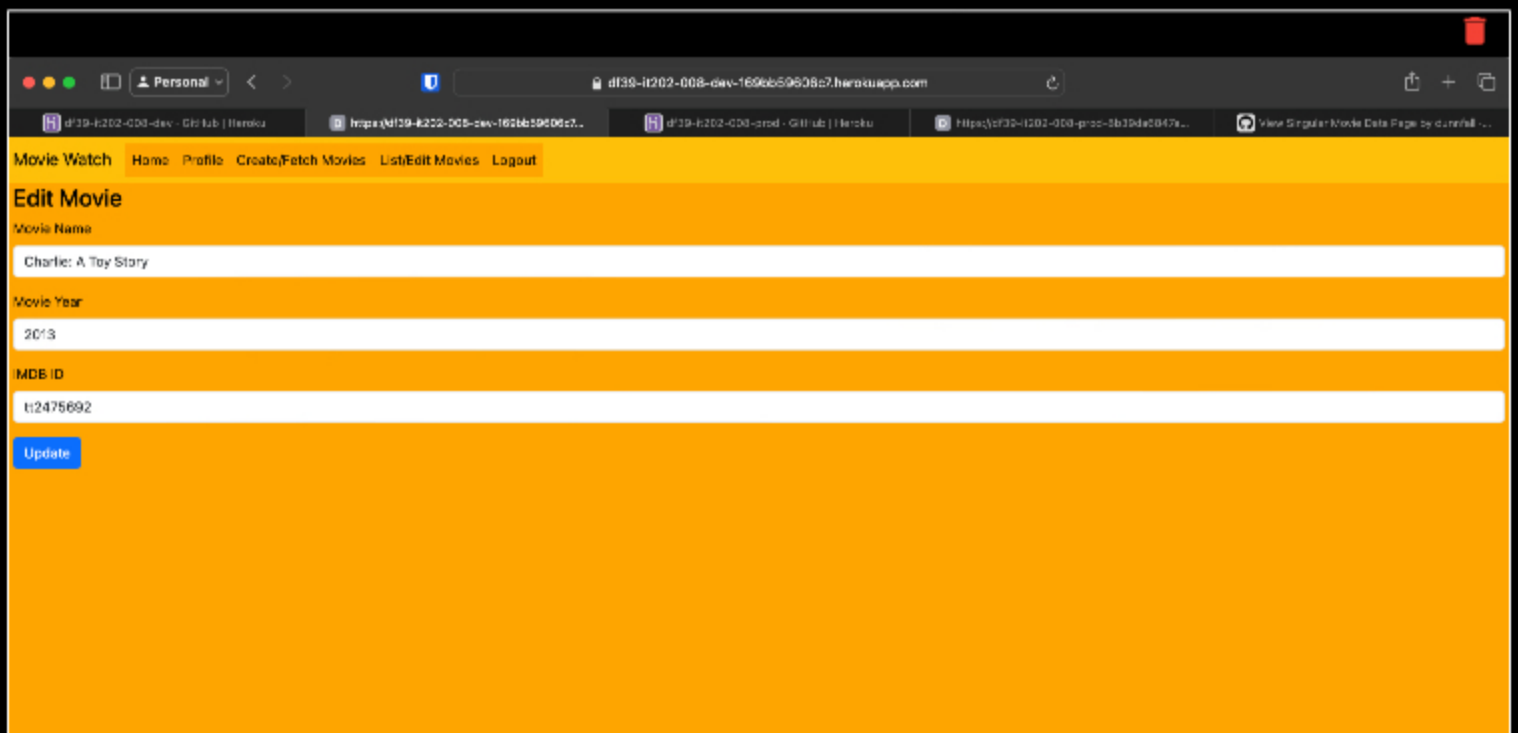
Checklist *The checkboxes are for your own tracking

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

Task Screenshots:

Gallery Style: Large View

Small Medium Large



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

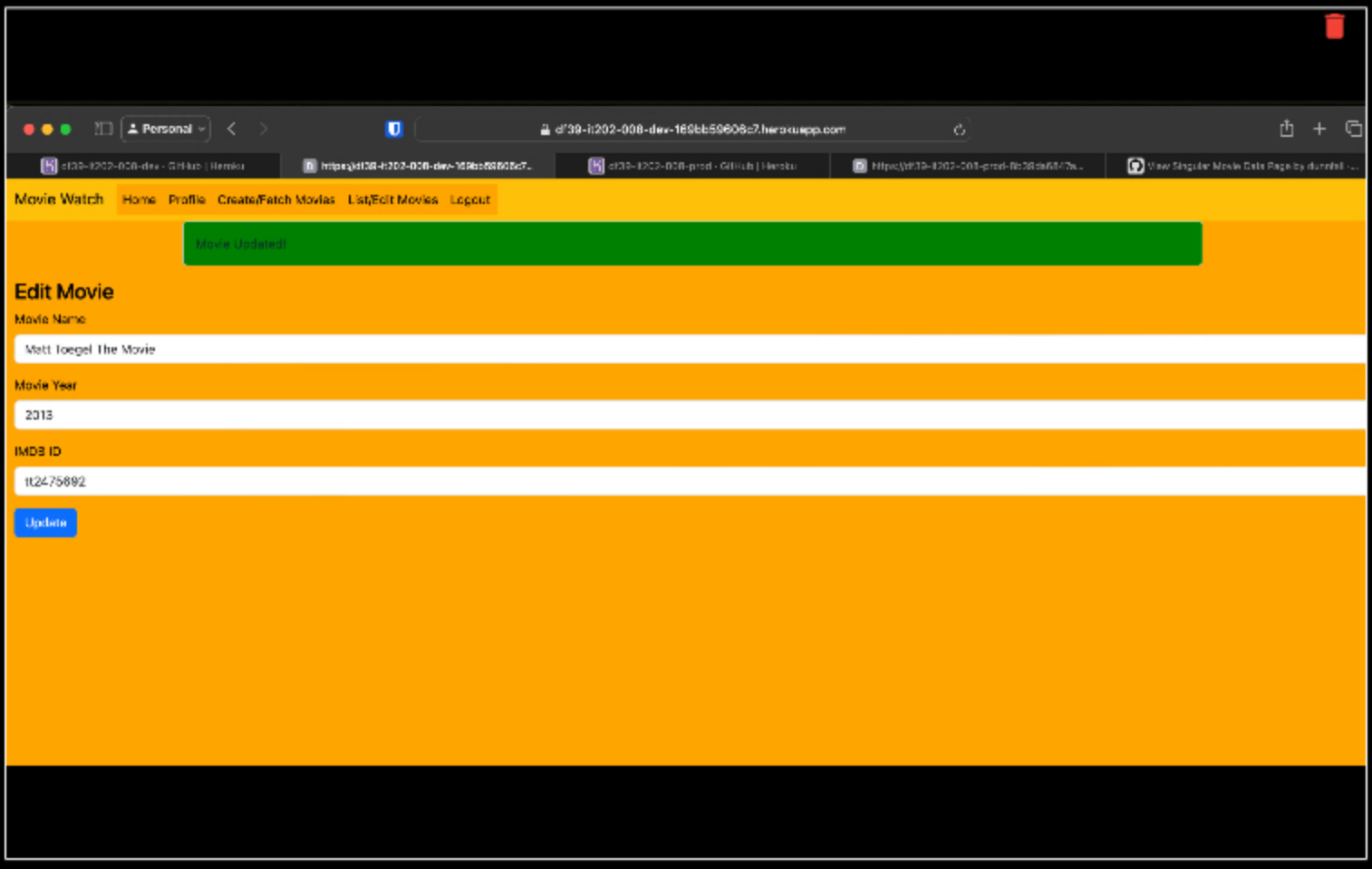
Checklist Items (4)

#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

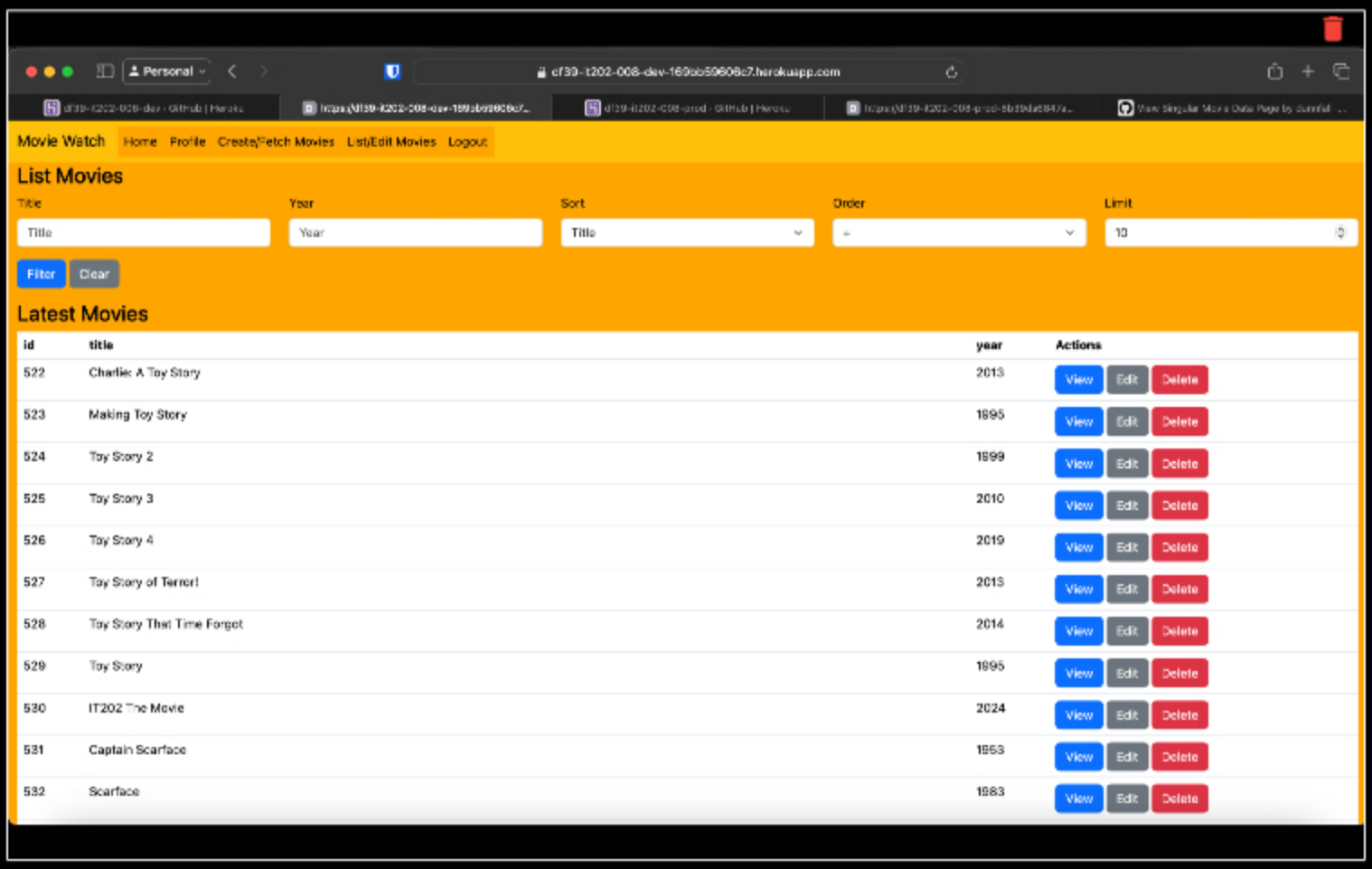
Checklist Items (4)

#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

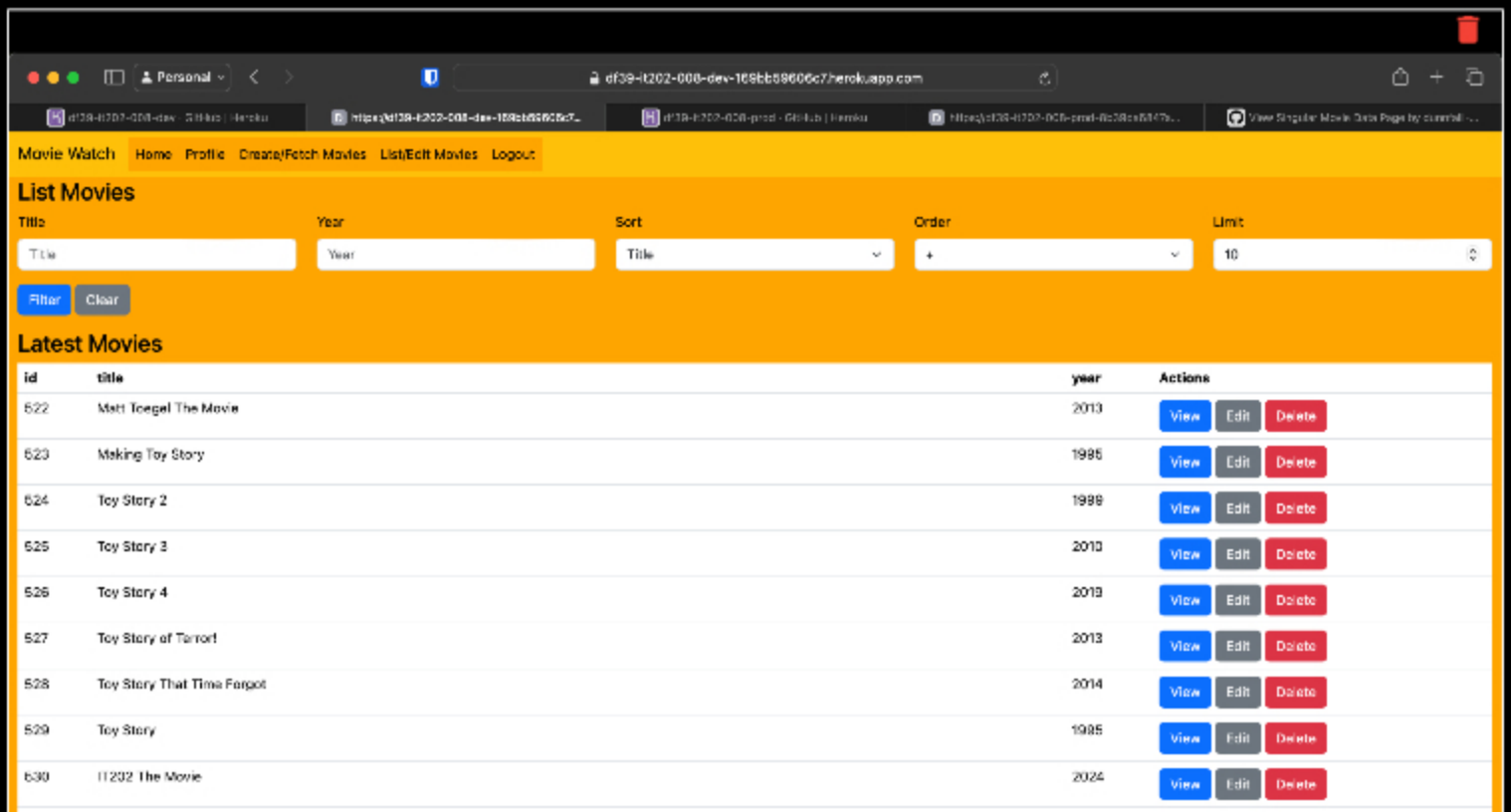
#6 Clearly caption screenshots



Before edit of data on list view page

Checklist Items (1)

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



031	Captain Scarface	1953	View Edit Delete
032	Scarface	1983	View Edit Delete

After of title being changed

Checklist Items (1)

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

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of edit page code

Checklist

*The checkboxes are for your own tracking

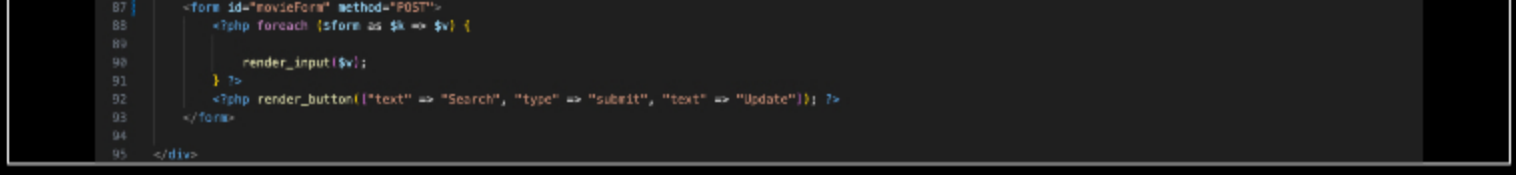
#	Points	Details
#1	1	Form should have correct data types for each property being requested
#2	1	Form should have correct validation for each field (HTML, JS, and PHP)
#3	1	Successful edit should have a user-friendly message
#4	1	Any errors should have user-friendly messages
#5	1	Include any other rules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

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```php
}
//DF39 4/10/2024
$stock = [];
if ($id > -1) {
    //fetch
    $db = getDB();
    $query = "SELECT title, year, imdb_id FROM `Movies` WHERE id = :id";
    try {
        $stmt = $db->prepare($query);
        $stmt->execute([":id" => $id]);
        $r = $stmt->fetch();
        if ($r) {
            $stock = $r;
        }
    } catch (PDOException $e) {
        error_log("Error fetching record: " . var_export($e, true));
        flash("Error fetching record", "danger");
    }
} else {
    flash("Invalid id passed", "danger");
    die(header("Location: " . get_url("admin/list_movies.php")));
}
$form = [];
if ($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);

    foreach ($form as $k => $v) {
        if (in_array($v["name"], $keys)) {
            $form[$k]["value"] = $stock[$v["name"]];
        }
    }
}


?>
<div class="container-fluid">
    <h3>Edit Movie</h3>
```

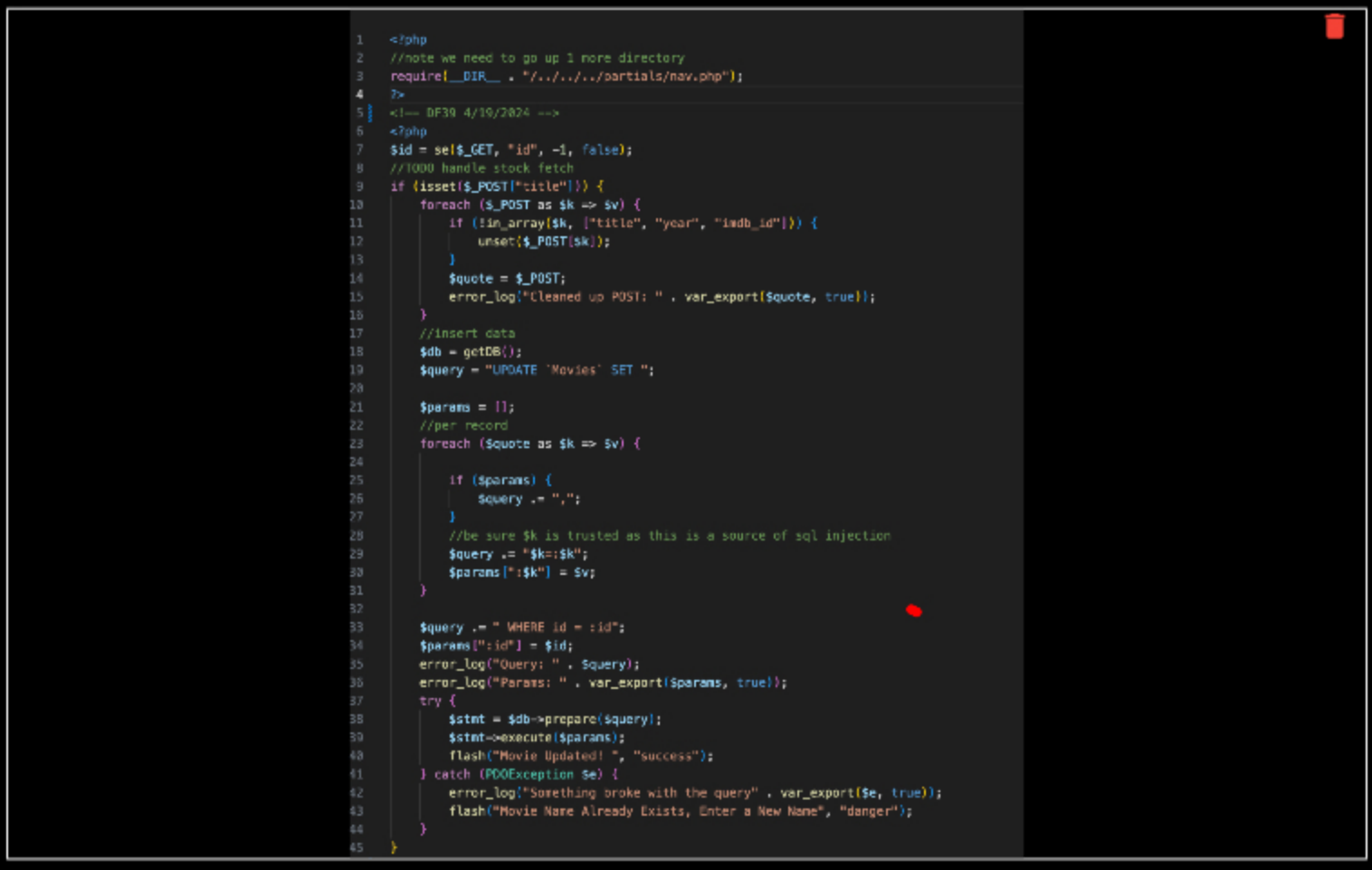


```
    <form id="movieForm" method="POST">
        <?php foreach ($form as $k => $v) {

            render_input($v);
        } ?>
        <?php render_button(["text" => "Search", "type" => "submit", "text" => "Update"]); ?>
    </form>

</div>
```

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



```
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");
?>
<!-- DF39 4/19/2024 -->
<?php
$id = se($_GET, "id", -1, false);
//TODO handle stock fetch
if (isset($_POST["title"])) {
    foreach ($_POST as $k => $v) {
        if (!in_array($k, ["title", "year", "imdb_id"])) {
            unset($_POST[$k]);
        }
        $quote = $_POST;
        error_log("Cleaned up POST: " . var_export($quote, true));
    }
    //insert data
    $db = getDB();
    $query = "UPDATE `Movies` SET ";

    $params = [];
    //per record
    foreach ($quote as $k => $v) {

        if ($params) {
            $query .= ",";
        }
        //be sure $k is trusted as this is a source of sql injection
        $query .= "$k=:$k";
        $params[":$k"] = $v;
    }

    $query .= " WHERE id = :id";
    $params[":id"] = $id;
    error_log("Query: " . $query);
    error_log("Params: " . var_export($params, true));
    try {
        $stmt = $db->prepare($query);
        $stmt->execute($params);
        flash("Movie Updated! ", "success");
    } catch (PDOException $e) {
        error_log("Something broke with the query" . var_export($e, true));
        flash("Movie Name Already Exists, Enter a New Name", "danger");
    }
}
```

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

Checklist Items (0)

```
//DF39 4/19/24
$form = [];
if (sstock) {
    $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)

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of records from DB

Checklist

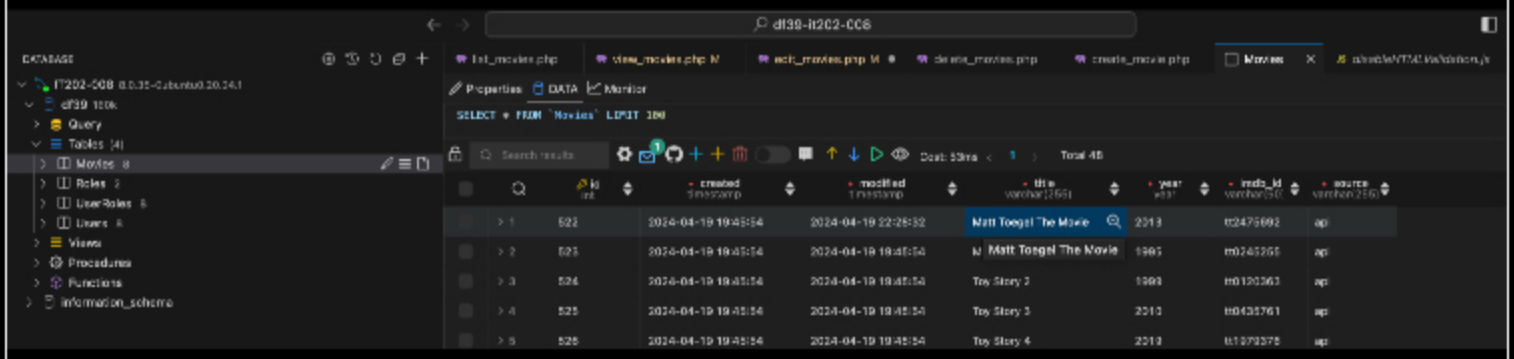
*The checkboxes are for your own tracking

#	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



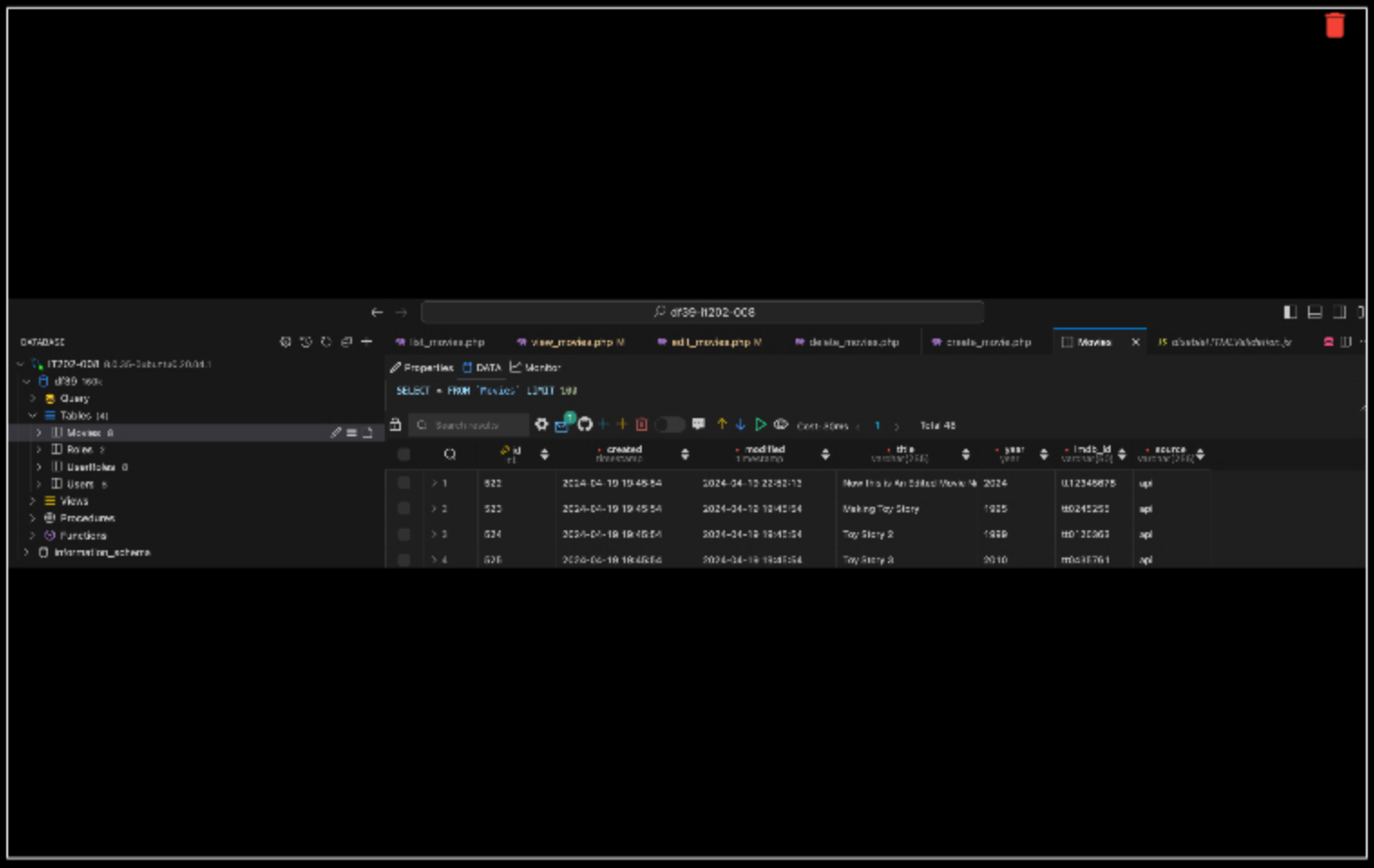
This screenshot shows Matt Toegel the Movie Record in DB before changing, title, imdb id and year

Checklist Items (3)

#1 Show a before and after screenshot of the record

#2 Note what differs

#3 Clearly caption screenshots



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

COLLAPSE

Task #4 - Points: 1

Text: Explain the process

Checklist *The checkboxes are for your own tracking

#	Points	Details
#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
#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 tiltle, 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.

COLLAPSE

Task #5 - Points: 1

Text: Add related links

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#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

COLLAPSE

Delete Handling (1 pt.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshots related to delete

Checklist *The checkboxes are for your own tracking

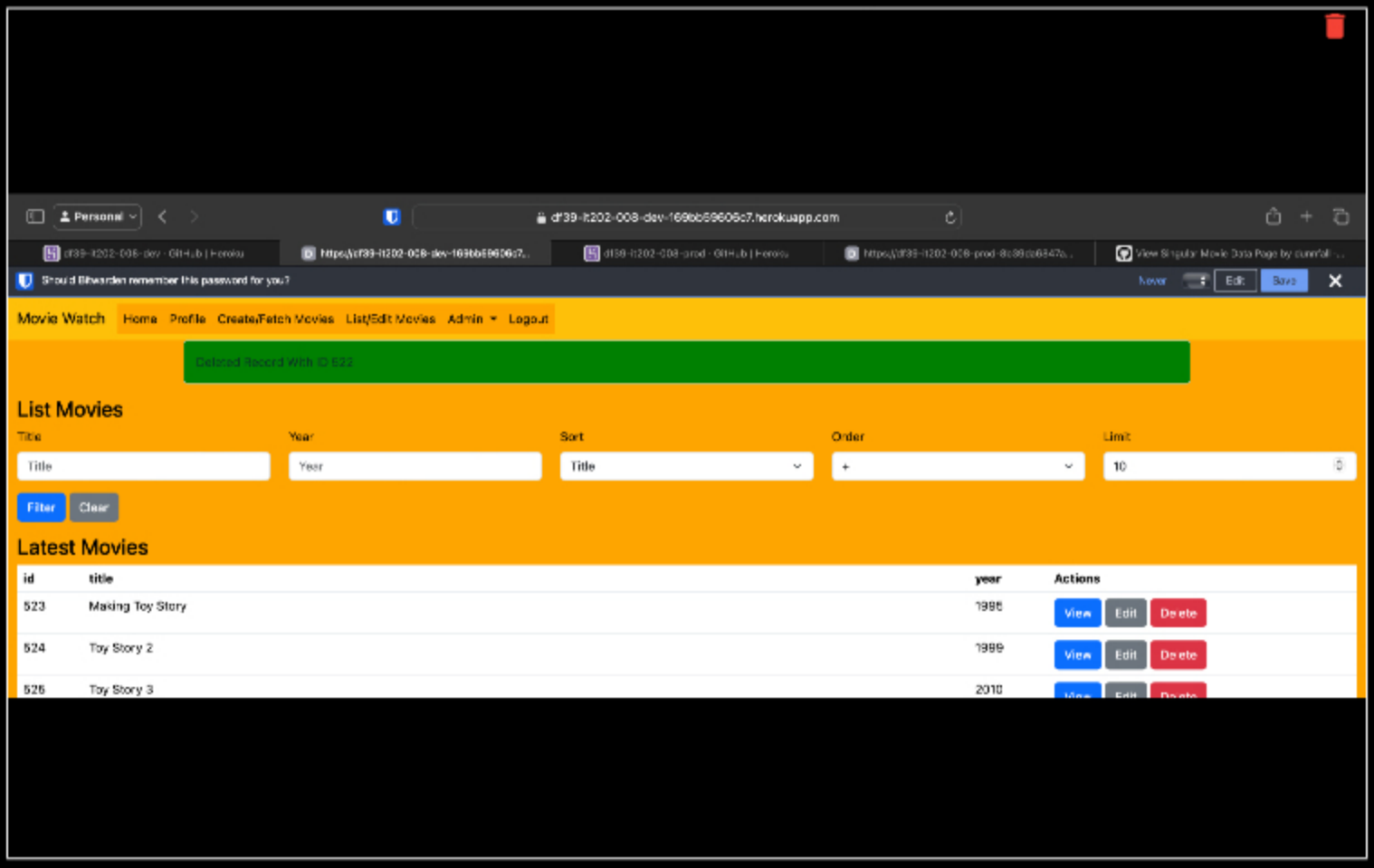
#	Points	Details

#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

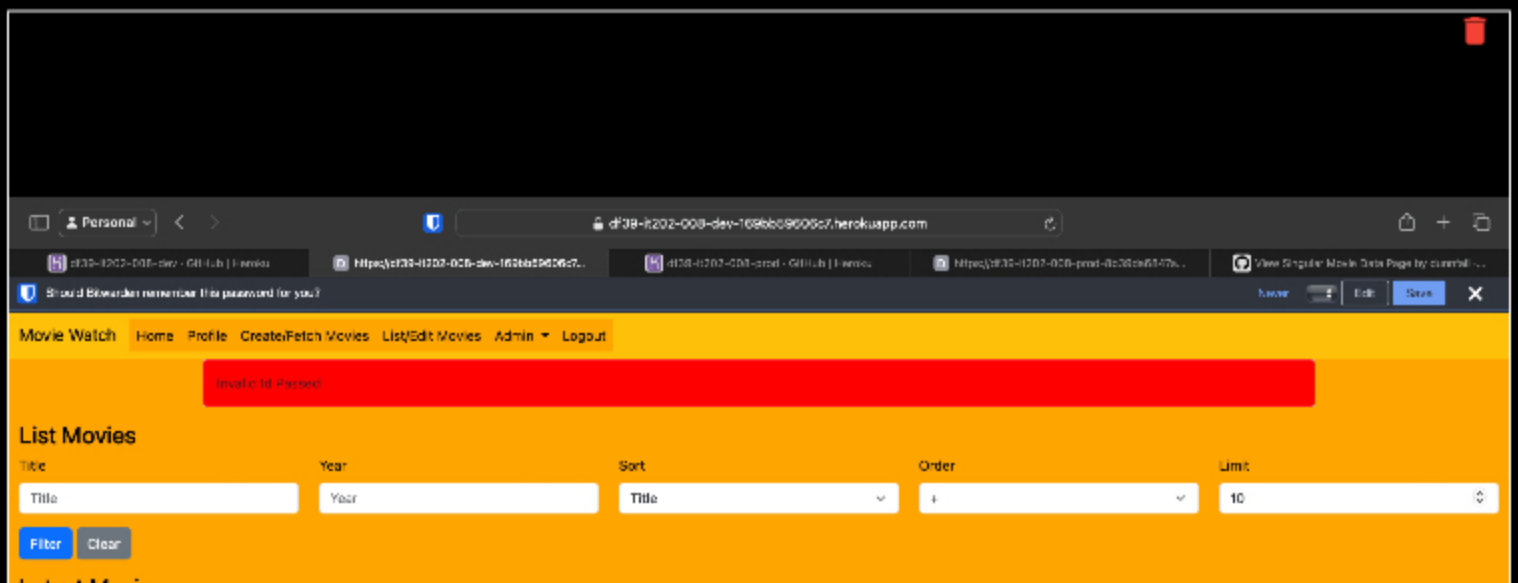
Small Medium Large

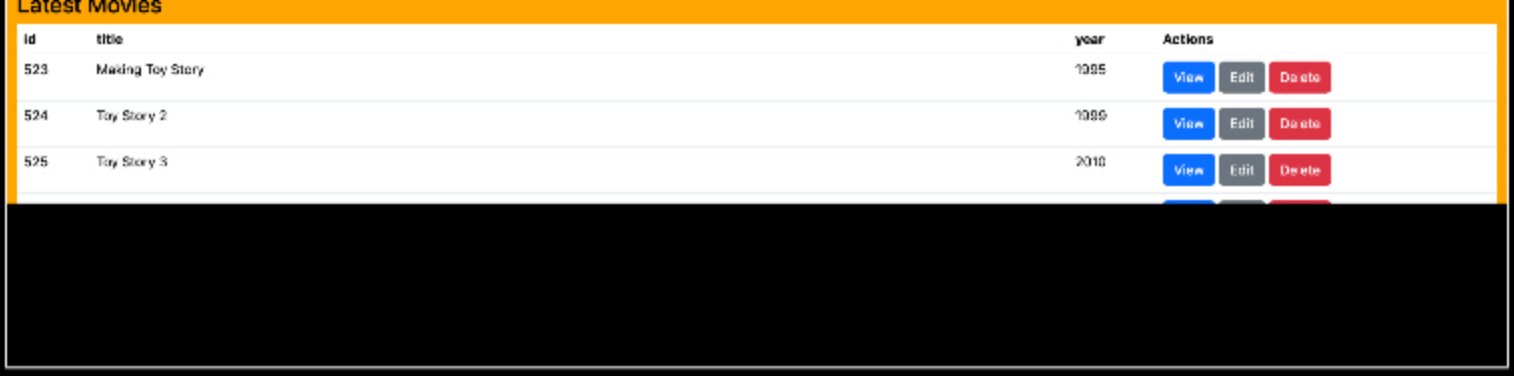


Success delete message with id.

Checklist Items (1)

#1 Show the success message of a delete





This error message appears when no id is passed.

Checklist Items (0)

```php
<?php

//DF39 4/19/2024
session_start();
require(__DIR__ . "/../../../lib/functions.php");

// require(__DIR__ . "/../../../lib/nav.php");
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("admin/list_movies.php")));
}


$id = se($_GET, "id", -1, false);
if ($id < 1) {
    flash("Invalid Id Passed", "danger");
    die(header("Location: " . get_url("admin/list_movies.php")));
}

$db = getDB();
$query = "DELETE FROM `Movies` WHERE id = :id";
try {
    $stmt = $db->prepare($query);
    $stmt->execute([":id" => $id]);
    flash("Deleted Record With ID $id", "success");
} catch (Exception $e) {
    error_log("Error Deleting Movie $id" . var_export($e, true));
    flash("Error Deleting Movie", "danger");
}
die(header("Location: " . get_url("admin/list_movies.php")));


```

Code of deleting movie

Checklist Items (1)

#3 Show the code related to the delete processing

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the data

Checklist

*The checkboxes are for your own tracking

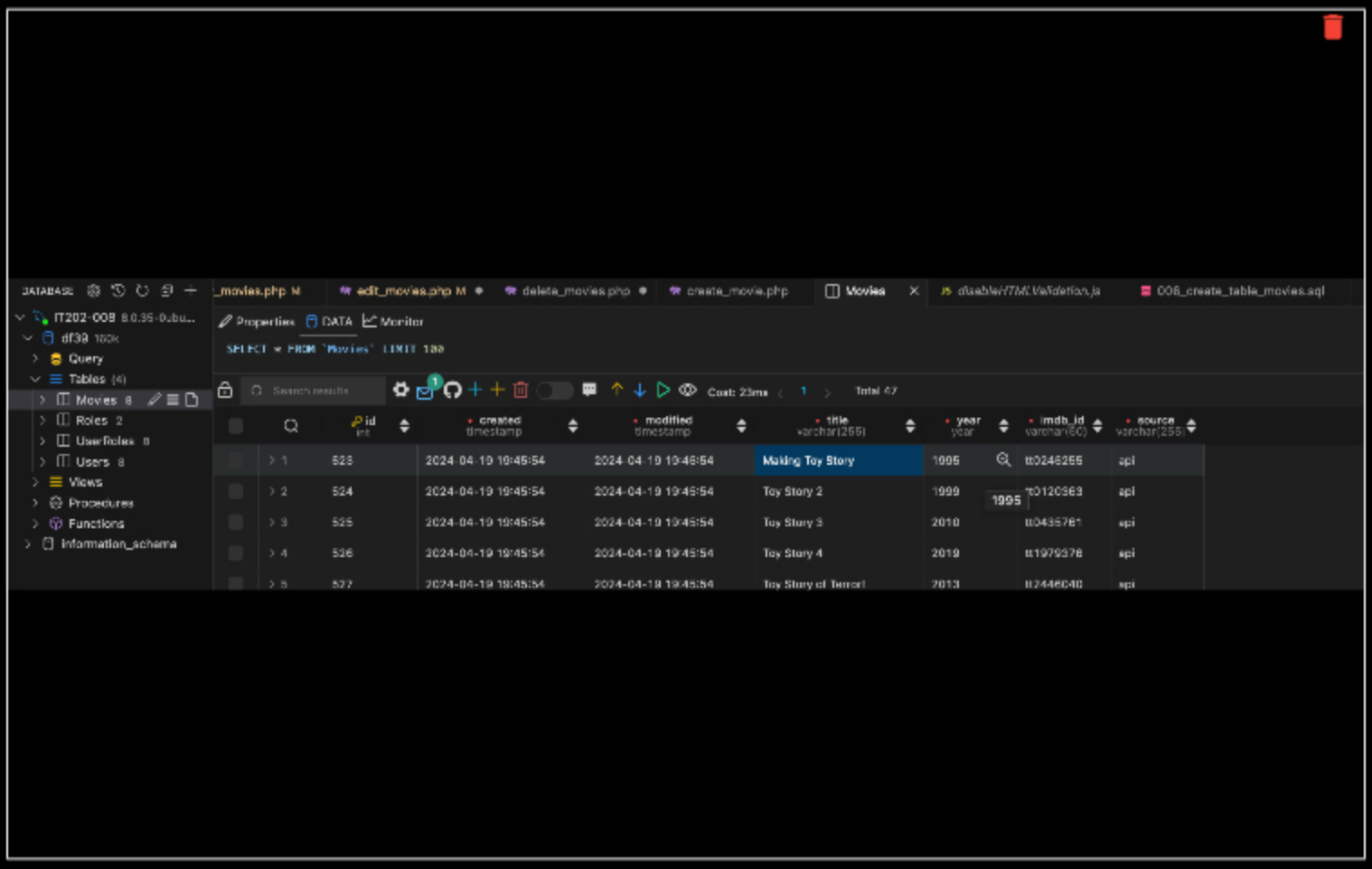
#	Points	Details
#1	1	Show a before and after screenshot of the DB data

#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

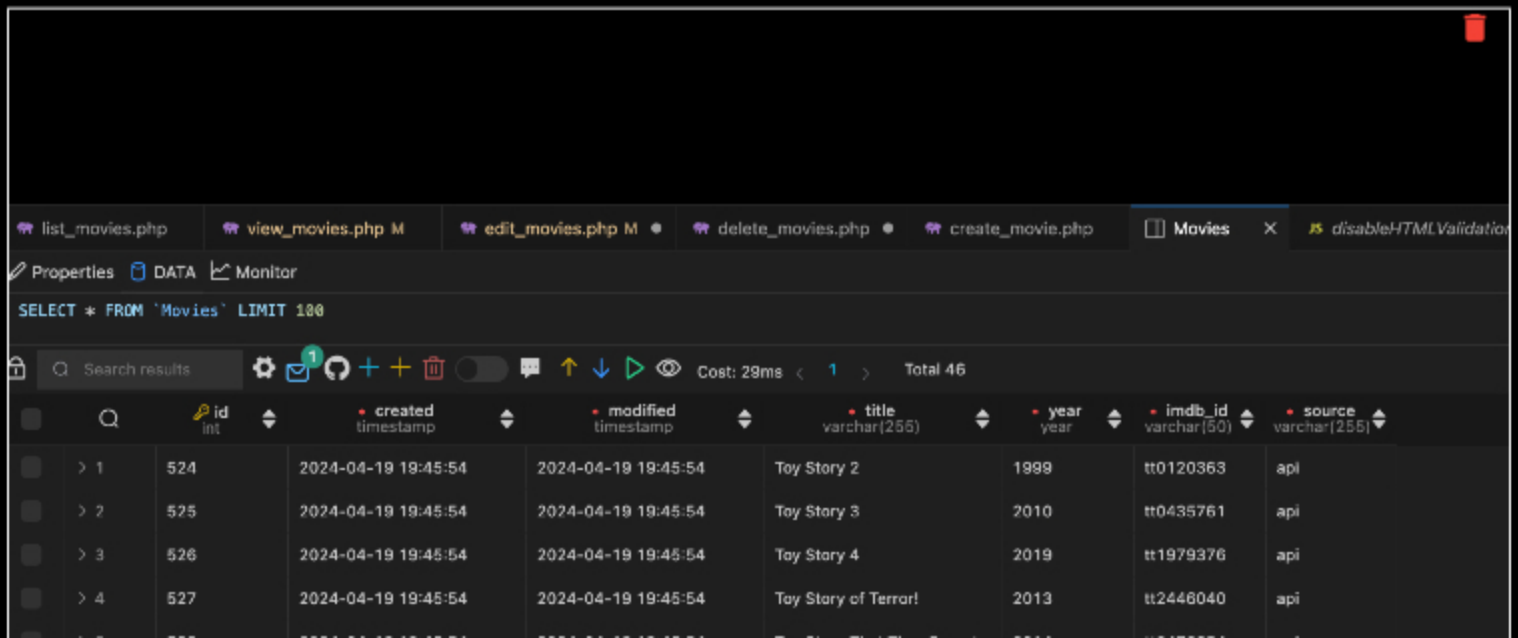
Small Medium Large



Making Toy Story movie shown in db before hard deleting

Checklist Items (1)

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



> 5	528	2024-04-19 19:45:54	2024-04-19 19:45:54	Toy Story That Time Forgot	2014	tt3473854	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

COLLAPSE

Task #3 - Points: 1

Text: Explain the delete logic

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Is it a soft or hard delete
#2	1	Are there any necessary roles or restrictions? (can only delete their data, can only be done by admin, etc)
#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

COLLAPSE

Task #4 - 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

URL #1
https://github.com/dunnfall/df39-it202-008/pull/55

Misc (1 pt.)

COLLAPSE

COLLAPSE

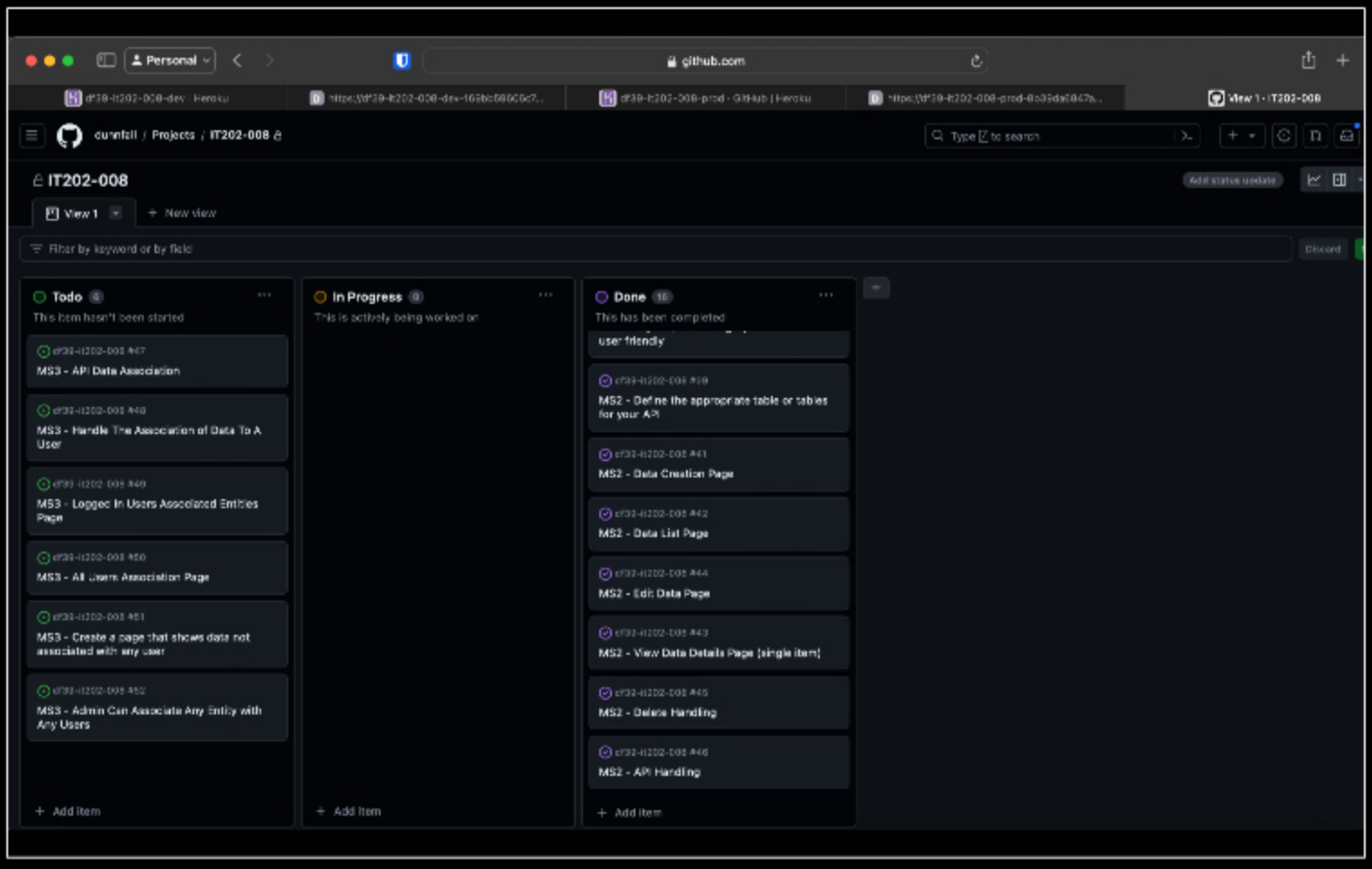
Task #1 - Points: 1

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

COLLAPSE

Task #2 - Points: 1

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

URL #1

https://github.com/users/dunnfall/projects/1

COLLAPSE

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.

COLLAPSE

Task #4 - Points: 1

Text: WakaTime Screenshot

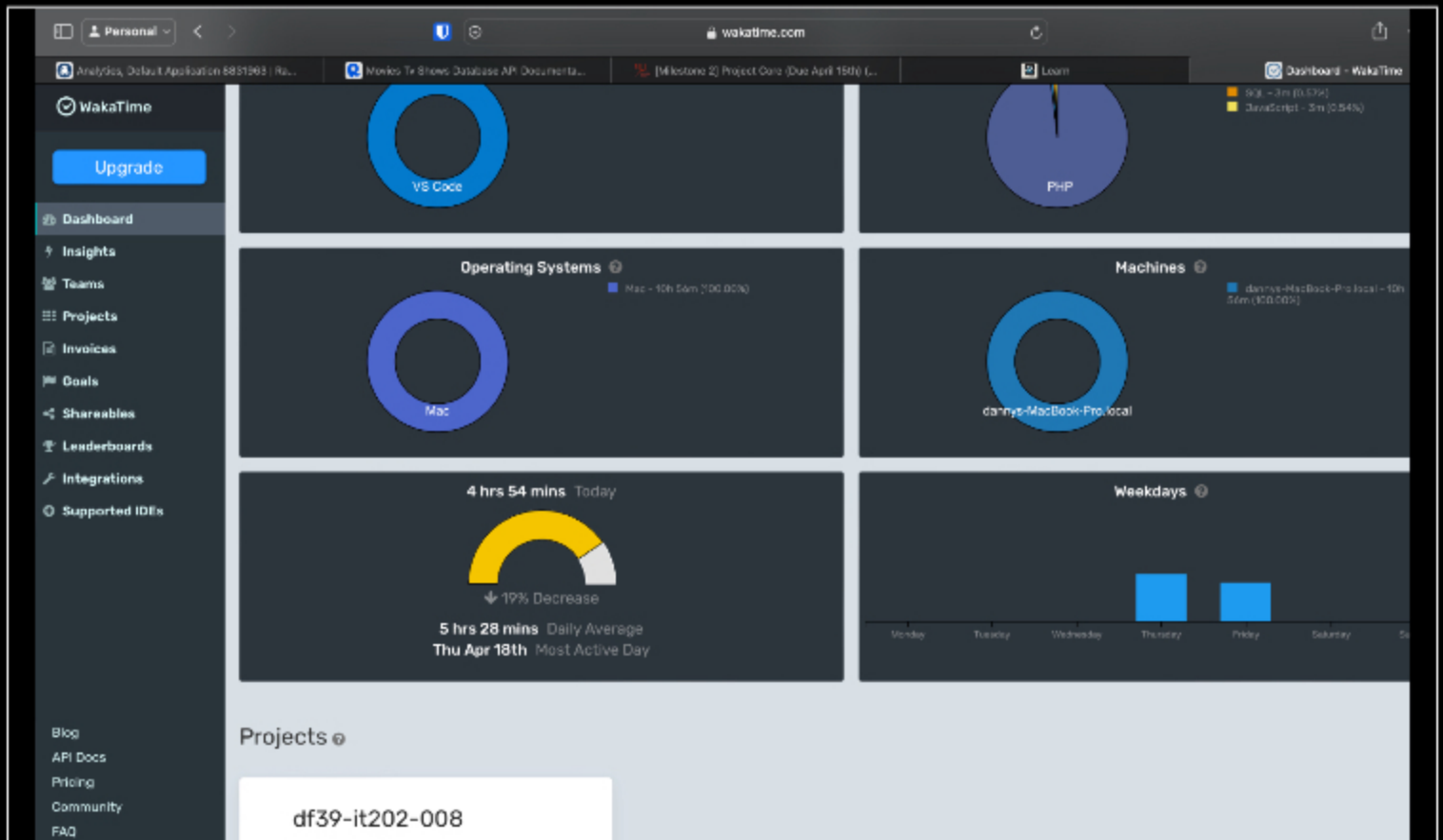
Details:

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



Wake time screenshot

End of Assignment