Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-api-project-milestone-2-2024-m24/grade/jd755

Course: IT202-007-F2024
Assigment: [IT202] API Project Milestone 2 2024 M24
Student: Jeremy R. (jd755)

Submissions:

Submission Selection

1 Submission [submitted] 11/21/2024 11:42:29 PM

Instructions

COLLAPSE

Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1XE96a8DQ52Vp49XACBDTNCq0xYDt3kF29cO88EWVwfo/view 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

Group

Group: Define Appropriate Tables for Data
Tasks: 2
Points: 1

COLLAPSE

Task

Group: Define Appropriate Tables for Data
Task #1: Screenshots of Table SQL
Weight: ~50%
Points: ~0.50

Points: ~0.50

COLLAPSE

Checklist *The checkboxes are for your own tracking

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

Sub-Task

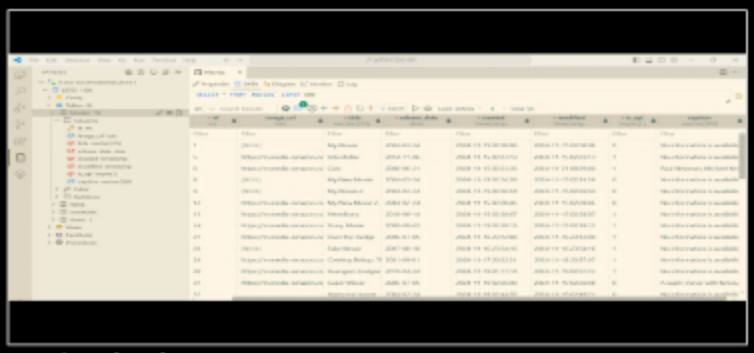
100%

Group: Define Appropriate Tables for Data
Task #1: Screenshots of Table SQL
Sub Task #1: Screenshot of the table (you can duplicate this subtask as needed)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Movies database

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain the columns and what data they represent (briefly), also note any normalization that may have been necessary

Response:

The image_url column holds the url for the movie poster. The title colums holds the title of the movie. The release_date column holds the release_date of the movie. The caption column holds the caption of the image_url.

End of Task 1

Task

100%

Group: Define Appropriate Tables for Data
Task #2: Add the pull request link for the branch related to this feature
Weight: ~50%
Points: ~0.50

COLLAPSE

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

Task URLs

URL #1
https://github.com/legitdaylight/jd755-IT202-007/pull/47

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/54

URL
https://github.com/legitdaylight/jd755-IT202-007

End of Task 2

End of Group: Define Appropriate Tables for Data
Task Status: 2/2

Group

100%

Group: Data Creation Page
Tasks: 4
Points: 2

COLLAPSE

Task

100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Weight: ~25%
Points: ~0.50

COLLAPSE

Details:
Heroku dev url must be visible in all relevant screenshots

Columns: 1

Sub-Task

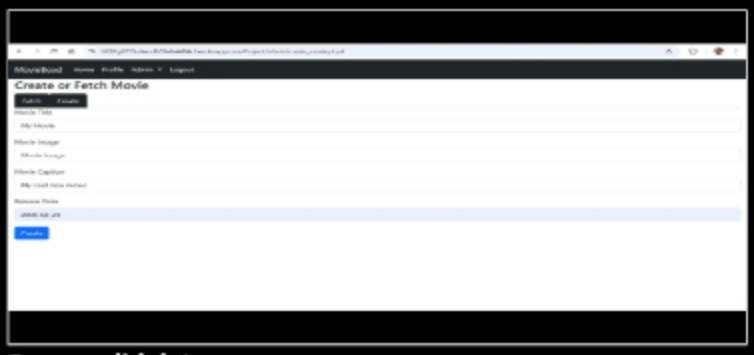
100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Sub Task #1: Show potentially valid data filled in for the custom creation page

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Some valid data

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Sub Task #2: Show how the API data is fetched for API data (must be server-side)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
#jd755 11/20/24
function fetch_movie($title)
{
    $data = ["exact"=>"false", "titleType"=>"movie"];
    $title = rawurlencode($title);
    $endpoint = "https://moviesdatabase.p.rapidapi.com/titles/search/title/$title";
    $isRapidAPI = true;
    $rapidAPIHost = "moviesdatabase.p.rapidapi.com";

    $result = get($endpoint, "STOCK_API_KEY", $data, $isRapidAPI, $rapidAPIHost);
    $output = [];
    if (se($result, "status", 400, false) == 200 && isset($result["response"])) {
        $result = json_decode($result["response"], true);
```

Screenshot 1

Screenshot 2

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the code

Response:

I start by creating an associative array called $data that holds all the options that I will use for my endpoint. In this case, the options were 'exact', which I set to false, and 'titleType', which I set to 'movie'. 'exact' refers to how strict the input title has to be. For example, If I send the request for a movie called "Cars" having the exact option set to true would only return the movie "Cars". However, with this option set to false the API will return multiple movies with the word 'cars' in its name. 'titleType' refers to what type of content the API will return. Since I only want movies, I set it to 'movie'. I then clean up the title using rawurlenclode() function, create the endpoint link, then define a variable called $result that holds the contents of the get() function with the endpoint, which calls the API. If the API was successful, I use a for loop to loop through all the arrays returned by the API and I extract the information I need for my database.

Sub-Task

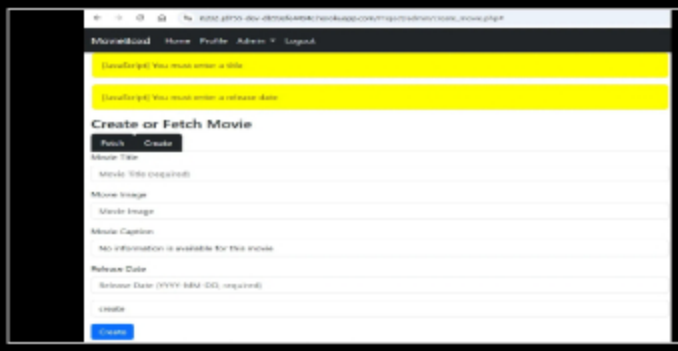
100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Sub Task #3: Show examples of validation messages

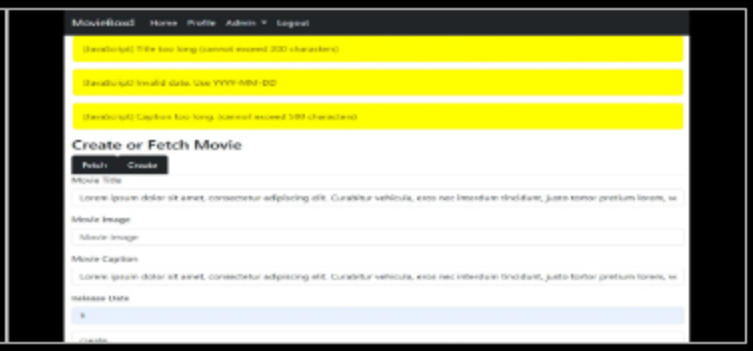
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Empty title and release data fields.



Title too long, caption too long, invalid date

Trying to create a movie that already exists in the database

Trying to fetch movie that isnt in the API

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

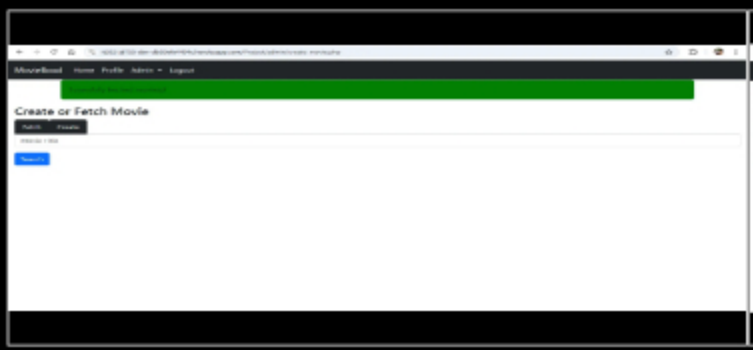
100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Sub Task #4: Show an example of successful creation message

Task Screenshots

Gallery Style: 2 Columns

4 2 1



successful fetch

Successful movie creation

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Data Creation Page
Task #1: Screenshots of the creation page
Sub Task #5: Design/Style should be considered (i.e., bootstrap, custom css, etc)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The fetch tab

The create tab

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain your design choices

Response:

The fetching from API and creation of data is split into two tabs to allow this page to be more organized. Teach section just consists of one or more input boxes with a button at the bottom for submition

End of Task 1

Task

100%

Group: Data Creation Page
Task #2: Screenshots of creation page code
Weight: ~25%
Points: ~0.50

COLLAPSE

Details:

Include ucid/date comments for each code screenshot

Columns: 1

Sub-Task

Sub-Task

100%

Group: Data Creation Page
Task #2: Screenshots of creation page code
Sub Task #1: Form should have correct data types for each property being requested

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Forms for data creation

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly talk about each field type and why it was chosen

Response:

For both fetch and create, the forms are all of the type text. There wasn't a need for anything else as the data I need are strings.

Sub-Task

100%

Group: Data Creation Page
Task #2: Screenshots of creation page code
Sub Task #2: Form should have correct validation for each field (HTML, JS, and PHP)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

JavaScript validation

PHP validation

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the validations

Response:

The validations for both JavaScript and PHP are checking for the same things.

1. Title cannot be blank and it cannot exceed 200 characters
2. The caption cannot exceed 500 characters
3. The release data cannot be blank and it must follow the xxxx-xx-xx data format. I used regex to check for this.

Sub-Task

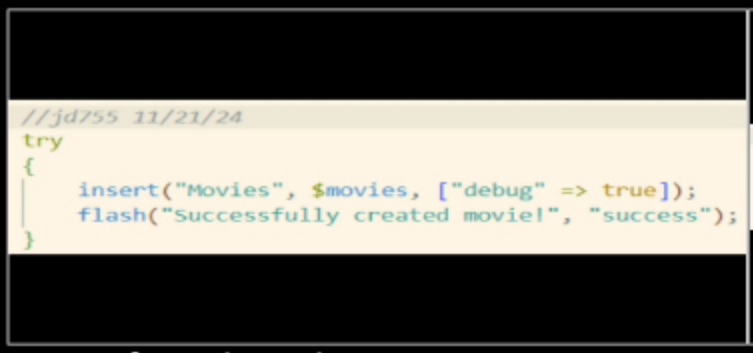
100%

Group: Data Creation Page
Task #2: Screenshots of creation page code
Sub Task #3: Successful creation should have a user-friendly message

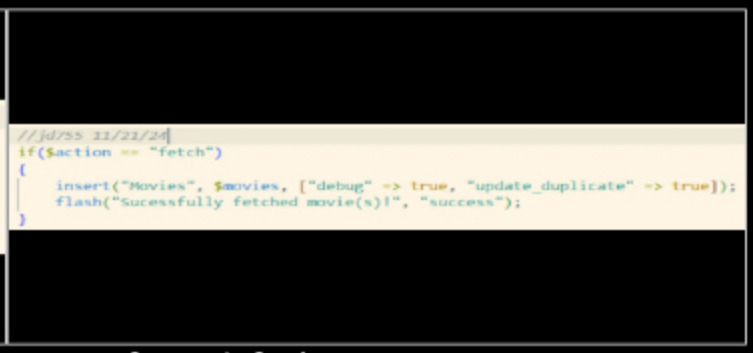
Task Screenshots

Gallery Style: 2 Columns

4 2 1



message for movie creation



Message for movie fetch

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain how duplicate/existing data is handled

Response:

Fetch: When the data is being fetched from the API, any duplicate data is replaced with the new information recieved. This is necessary as there can easily be duplicates if keywords from different movies overlap. Create: After the form is sent to the database, it checks if the title and the release data combination already exists. If there is a duplicate, a warning is sent to the user and nothing is sent to the database.

Sub-Task

100%

Group: Data Creation Page
Task #2: Screenshots of creation page code
Sub Task #4: Any errors should have user-friendly messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
#jd755 11/21/24 <- #87-124 if(count($movies) > 0)
else
{
    flash("[PHP] Movie could not be found.", "warning");
}
```

Fetch error if a movie wasn't found

```
<?php
#jd755 11/21/24
function movie_check_duplicate($errorinfo)
{
    if ($errorinfo[1] === 1062) {
        //https://www.php.net/manual/en/function.preg-match.php
        preg_match("/Movies.(\w+)/", $errorinfo[2], $matches);
        if (isset($matches[1])) {
            flash("ihe chosen movie is already in the database.", "warning");
        } else {
            flash("An unhandled error occured", "danger");
            //this will log the output to the terminal/console that's running the php server
            error_log(var_export($errorinfo, true));
        } <- #9-13 else
    } else {
        flash("An unhandled error occured", "danger");
        //this will log the output to the terminal/console that's running the php server
        error_log(var_export($errorInfo, true));
    } <- #14-18 else
} <- #3-19 function movie_check_duplicate($errorInfo)
```

function that checks if there is any duplicates

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly describe the scenarios

Response:

If the API couldn't find the movie with the submitted title, it displays a message to the user "Movie cannot be found". If the movie that was created already exists in the database, it displays a message to the user "Chosen movie already exists"

Sub-Task

100%

Group: Data Creation Page

Task #2: Screenshots of creation page code

Sub Task #5: Include the form/process for fetching API data

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
//jd755 11/21/24
if ($action === "fetch")
{
    $result = fetch_movie($title);
    error_log("Data from API" . var_export($result, true));
    if ($result)
    {
        $movies = $result;
    }
} <- #31-39 if ($action === "fetch")
```

Code for fetching the movie from the API

```
//jd755 11/21/24
if($action == "fetch")
{
    insert("Movies", $movies, ["debug" => true, "update_duplicate" => true]);
    flash("Sucessfully fetched movie(s)!", "success");
}
```

Putting the API data into the database

form used to get the title for the API call

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the steps (include how duplicates are handled)

Response:

First, the user enters the title into the form. Then, the title is sent to the function fetch_movie(), where the API call is made. This function returns an array which is sent to $result. $result is checked to see if it has any data or not. If it does, set $movies equal to $result. Then, call the insert() function with $movies and with "update_duplicate" set to true. This is to prevent any errors from the database about any duplicate values.

Sub-Task

100%

Group: Data Creation Page

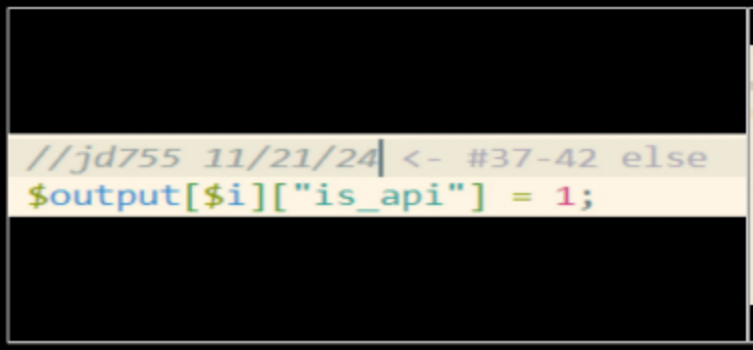
Task #2: Screenshots of creation page code

Sub Task #6: Include some indicator between custom data and API data

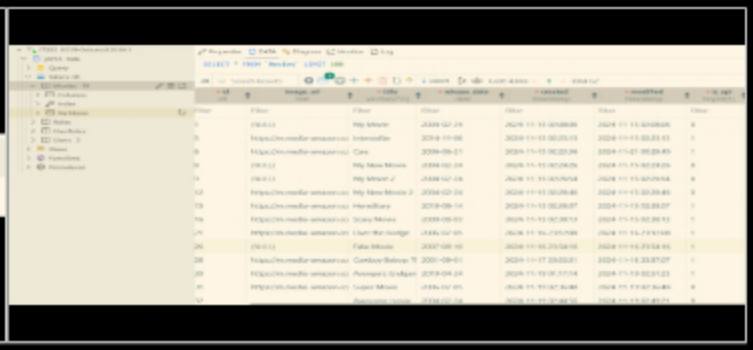
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Inside of movie_api.php. This column is added to indicate that this data is from the API.



is_api column from the database

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly mention what the indicator is (i.e., api_id if the API has ids or is_api as a boolean-like column, etc)

Response:

The indicator is is_api with a boolean-like column

Sub-Task

100%

Group: Data Creation Page

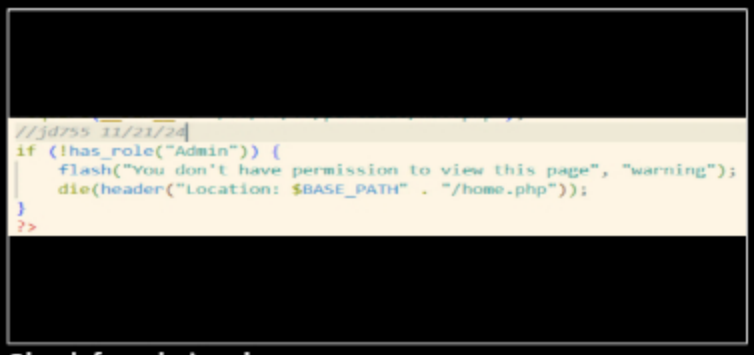
Task #2: Screenshots of creation page code

Sub Task #7: Include any other rules like role guards and login checks

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//jdr55 11/21/24
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}
?>
```

Check for admin role

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic/reasoning

Response:

Non-admin users will be using the database to get movies. As there is a limit on the amount of times the API can be called, I decided that non admins shouldn't have direct acess to it. Non-admins also aren't allowed to create movies themselves.

End of Task 2

Task

100%

Group: Data Creation Page
Task #3: Screenshot of records from DB
Weight: ~25%
Points: ~0.50

COLLAPSE

Columns: 1

Sub-Task

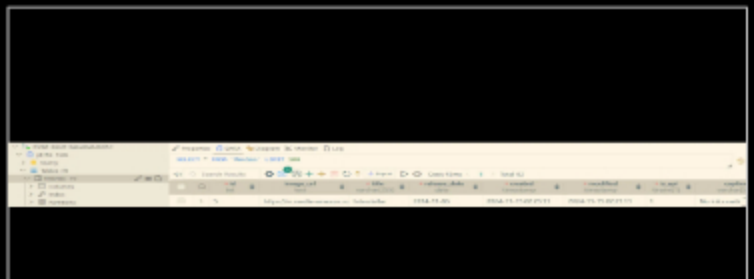
100%

Group: Data Creation Page
Task #3: Screenshot of records from DB
Sub Task #1: Show at least one record fetched from the API

Task Screenshots

Gallery Style: 2 Columns

4 2 1



record fetched from API

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (note what differs from a custom record)*

Sub-Task

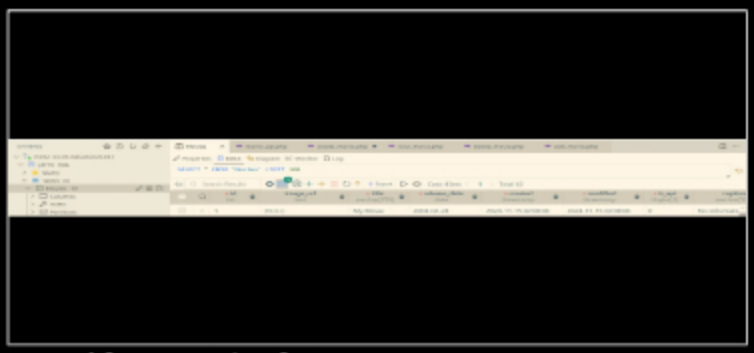
100%

Group: Data Creation Page
Task #3: Screenshot of records from DB
Sub Task #2: Show at least one record created via the creation form

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Record from creation form

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (note what differs from the API record)*

End of Task 3

Task

100%

Group: Data Creation Page
Task #4: Add related links
Weight: ~25%
Points: ~0.50

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Details
#1	Include the heroku prod link for this page
#2	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

Task URLs

URL #1
https://github.com/legitdaylight/jd755-IT202-007/pull/47

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #2

URL

https://github.com/legitdaylight/jd755-IT202-007/pull/54

https://github.com/legitdaylight/jd755-IT202-007

URL #3

https://it202-jd755-prod-028c8689138c.herokuapp.com/Project/admin/create_movie.php

https://it202-jd755-prod-028c8689138c.herokuap

End of Task 4

End of Group: Data Creation Page
Task Status: 4/4

Group

100%

Group: Data List Page (many entities)
Tasks: 3
Points: 2

COLLAPSE

Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Weight: ~33%
Points: ~0.67

COLLAPSE

Details:
Heroku dev url must be visible in all relevant screenshots

Columns: 1

Sub-Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #1: Show the page of your entities listed (have a reasonable number shown)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

List page

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly describe the page

Response:

The page has the filters at the top with the table at the bottom that has some data from the database.

Sub-Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #2: Show the filter/sort form based on your data and the required limit field

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The filter with the limit at the top as a warning

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly mention what's available to the user

Response:

The user has the ability to view, edit, or delete data from the buttons on the side. The table itself only has 4 pieces of information.

Sub-Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #3: Demonstrate a few varied filters/sorts

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Title: Cars Number of Records: 5

Title: Power Ranger Number of Records: 2

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #4: Demonstrate a filter that doesn't have any records (should show an appropriate message)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

title: random Number of records: left blank

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #5: 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, include examples from different user roles)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The table. This page is only accessible by admins

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Mention which users can interact with the view, edit, and delete links

Response:

As this page is only available to admins, the view, edit, and delete links are only accessable to them.

Sub-Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #6: Each list item should have a summary of the entity (likely won't be the entire entity data)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Summary of data

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Data List Page (many entities)
Task #1: Screenshots of the list page
Sub Task #7: Design/Style should be considered (i.e., bootstrap, custom css, etc)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Base list page

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain your design choices

Response:

The filters are at the top for easy reach, and the buttons are the side of each data to make it clear its associated with said data. A table was used to organize all the data from the database and present them in clear rows and columsns.

End of Task 1

Task

100%

Group: Data List Page (many entities)
Task #2: Screenshots of the list page code
Weight: ~33%
Points: ~0.67

COLLAPSE

Details:
Include ucid/date comments for each code screenshot

Columns: 1

Sub-Task

100%

Group: Data List Page (many entities)
Task #2: Screenshots of the list page code
Sub Task #1: Show the filter/sort form generation

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The form

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain how the code works (i.e, some stuff may be dynamic)

Response:

I create an array called $filterData that holds the current filter data (the title and filter number). I use this array in the variable $deleteURL. This is to create the first part of the delete_movie.php url. This is needed to maintain the values of the sort after the deletion is complete. I pass this value along with the data of the sort into the $table array. Within the div, I have 2 input boxes, one for the title and one for the filter number. After those, I have the submit button. Finally at the bottom, I pass the $table array into the render_table() function to create the table with all the data.

Sub-Task

100%

Group: Data List Page (many entities)

Task #2: Screenshots of the list page code

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Code handling the filtering

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the related logic

Response:

I grab the title and the filter from the URL using GET and set them to the variables $title and $num respectively. I then check if $num has a value. If it doesn't, it's set to the value 10 which is the default value for the filter. If it does have a value but the value is less than 1 or greater than 100, I send a warning and set the value of $num to 10. I then check if $title has a value. If it doesn't, I set the $query to just grab the latest 10 movies sorted by date of creation into the database. If title does have a value, I check if it is less than or equal to 200. Greater than 200 gives a warning and calls $query without a title variable. Otherwise, $query now includes the title.

Sub-Task

100%

Group: Data List Page (many entities)

Task #2: Screenshots of the list page code

Sub Task #3: Show how the output is generated and displayed

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The call to the database and the creation of the $table array.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the related logic

Response:

After the query is created, it is executed. If all goes well, the results are saved to the $results variable. Later, the $results variable is put into the $table array along side with the name of the table, and the urls for viewing, editing, and deletion. This $table variable is passed to the function render_table() where it forms the table and displays data row per row.

Sub-Task

100%

Group: Data List Page (many entities)

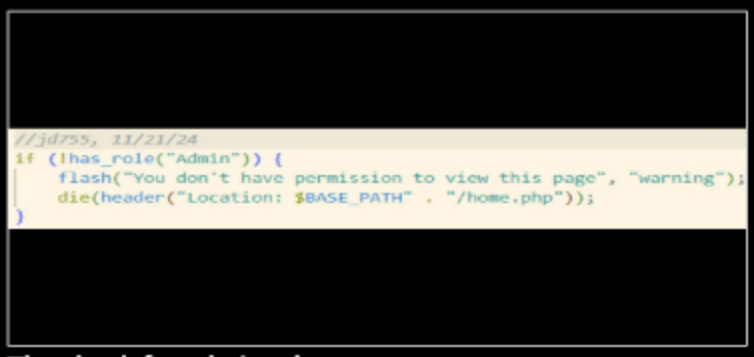
Task #2: Screenshots of the list page code

Sub Task #4: Show any restrictions like role guard or login checks

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```php
//jd755, 11/21/24
if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: $BASE_PATH" . "/home.php"));
}
```

The check for admin role

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Task Response Prompt

Briefly explain the logic/reasoning

Response:

As this page is for admins only, I have the check for the admin role.

End of Task 2

Task

100%

Group: Data List Page (many entities)
Task #3: Add related links
Weight: ~33%
Points: ~0.67

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Details
#1	Include the heroku prod link for this page
#2	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

Task URLs

URL #1
https://it202-jd755-prod-028c8689138c.herokuapp.com/Project/admin/list_movies.php?title=&filter=

URL
https://it202-jd755-prod-028c8689138c.herokuap

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/49

URL
https://github.com/legitdaylight/jd755-IT202-007

URL #3
https://github.com/legitdaylight/jd755-IT202-007/pull/55

URL
https://github.com/legitdaylight/jd755-IT202-007

End of Task 3

End of Group: Data List Page (many entities)
Task Status: 3/3

Group

100%

Group: View Details Page (single entity)
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Heroku dev url must be visible in all relevant screenshots

Columns: 1

Sub-Task

100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #1: Entity should be fetch by id (via the url)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The view page

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #2: A missing id should redirect back to the list page with an applicable message

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Invalid ID

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #3: Design/Style should be considered (i.e., bootstrap, custom css, etc)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The view movie page

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain your design choices

Response:

I centered all the information on the page to make it look cleaner. Underneath the image, I have the caption of the movie. I then have the release date. Right at the bottom I have the options to edit or delete (admin only).

Sub-Task 100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #4: Data shown should be more detailed/inclusive than the summary view

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Information on the view page

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the details shown and how it differs from the list view

Response:

The main differences from the view page vs the list page is that the caption and the release data is visible but the movie ID and is_api is not. The image is in both the list and view pages however the view page actually displays the image.

Sub-Task

100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #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)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Edit button in gray

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: View Details Page (single entity)
Task #1: Screenshots of the details page
Sub Task #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)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Delete button in red

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

End of Task 1

Task

100%

Group: View Details Page (single entity)
Task #2: Screenshots of the details page code
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Include ucid/date comments for each code screenshot

Columns: 1

Sub-Task

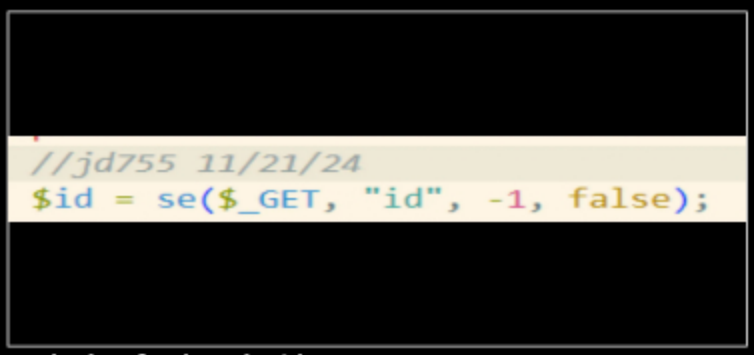
100%

Group: View Details Page (single entity)
Task #2: Screenshots of the details page code
Sub Task #1: Show how id is fetched

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Code that fetches the id

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic and how it's handled when the property is missing or not "valid"

Response:

The id is taken from the URL using the $_GET variable.

Sub-Task

100%

Group: View Details Page (single entity)
Task #2: Screenshots of the details page code
Sub Task #2: Show the DB query to get the record

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```
//id755 11/21/24
$movie = [];
if ($id > -1)
{
    //fetch
    $db = getDB();
    $query = "SELECT id, title, image_url, release_date, caption FROM `Movies` WHERE id = :id";
    try {
        $stmt = $db->prepare($query);
        $stmt->execute([":id" => $id]);
        $r = $stmt->fetch();
        if ($r) {
            $movie = $r;
        }
    } catch (PDOException $e) {
        error_log("Error fetching movie: " . var_export($e, true));
        flash("Error fetching record", "danger");
    }
} <- #11-26 if ($id > -1)
else
{
    flash("Invalid id passed", "danger");
    die(header("Location:" . get_url("admin/list_movies.php?title=&filter=")));
}
```

Code to get the data from the database

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

I check if the ID is valid. If it is not, I send a warning to the user. If it is valid, I create an SQL query fetching the required columns with the current ID

Sub-Task

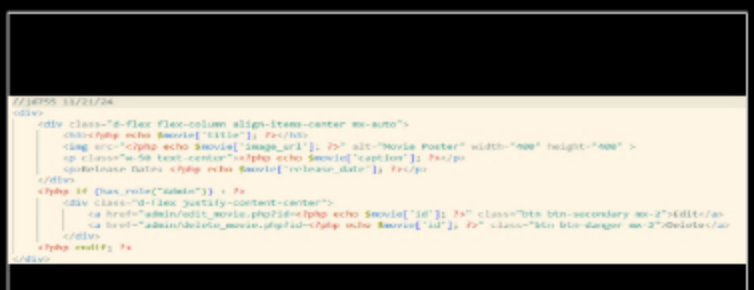
100%

Group: View Details Page (single entity)
Task #2: Screenshots of the details page code
Sub Task #3: Show the code related to presenting the data and showing the links

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```
//id755 11/21/24
<div>
    <div class="d-flex flex-column align-items-center mx-auto">
        <h1><?php echo $movie['title']; ?></h1>
        <img src="<?php echo $movie['image_url']; ?>" alt="Movie Poster" width="400" height="400" >
        <p class="w-50 text-center"><?php echo $movie['caption']; ?></p>
        <p>Release Date: <?php echo $movie['release_date']; ?></p>
    </div>
    <?php if (has_role("Admin")) : ?>
        <div class="d-flex justify-content-center">
            <a href="admin/edit_movie.php?id=<?php echo $movie['id']; ?>" class="btn btn-secondary mx-2">Edit</a>
            <a href="admin/delete_movie.php?id=<?php echo $movie['id']; ?>" class="btn btn-danger mx-2">Delete</a>
        </div>
    <?php endif; ?>
</div>
```

Code that displays all the information

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic

Response:

The data from the database is put inside the $movies variable. I start by getting the title of the movie and placing it into an h3 element. I then get the image of the movie and display it using the img element. I then place the caption and release data beneath the image using p elements. Underneath the above div, I placed the two buttons. I used a elements with the href set to their respective locations.

End of Task 2

Task

100%

Group: View Details Page (single entity)
Task #3: Add related links
Weight: ~33%
Points: ~0.33

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Details
#1	Include the heroku prod link for this page
#2	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

Task URLs

URL #1
https://it202-jd755-prod-028c8689138c.herokuapp.com/Project/view_movie.php?id=130

URL
https://it202-jd755-prod-028c8689138c.herokuap

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/53

URL
https://github.com/legitdaylight/jd755-IT202-007

End of Task 3

End of Group: View Details Page (single entity)
Task Status: 3/3

Group

100%

Group: Edit Data Page
Tasks: 4

Points: 2

COLLAPSE

Task

100%

Group: Edit Data Page
Task #1: Screenshots of the edit page
Weight: ~25%
Points: ~0.50

COLLAPSE

Details:
Heroku dev url must be visible in all relevant screenshots

Columns: 1

Sub-Task

100%

Group: Edit Data Page
Task #1: Screenshots of the edit page
Sub Task #1: Show before and after screenshots of data you'll edit (two screenshots required)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

before editing

After editing

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain what you changed
Response:

I changed the caption of the movie.

Sub-Task

100%

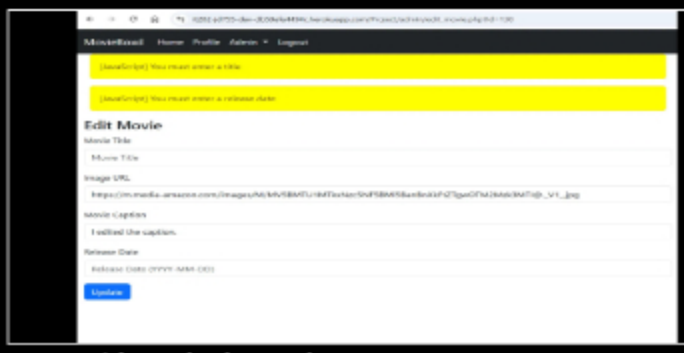
Group: Edit Data Page
Task #1: Screenshots of the edit page

Sub Task #2: Show examples of validation messages

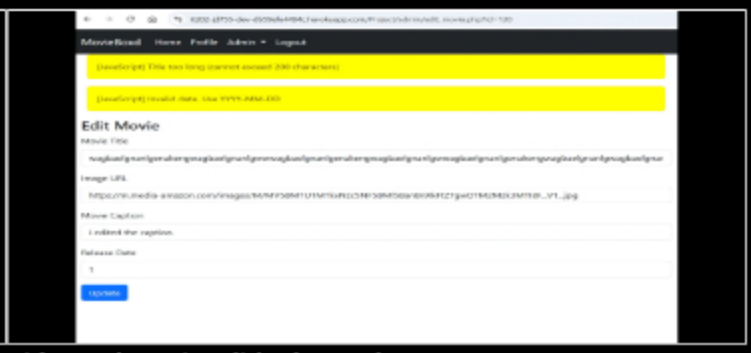
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Empty title and release date



Title too long, invalid release date

caption too long

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task

100%

Group: Edit Data Page

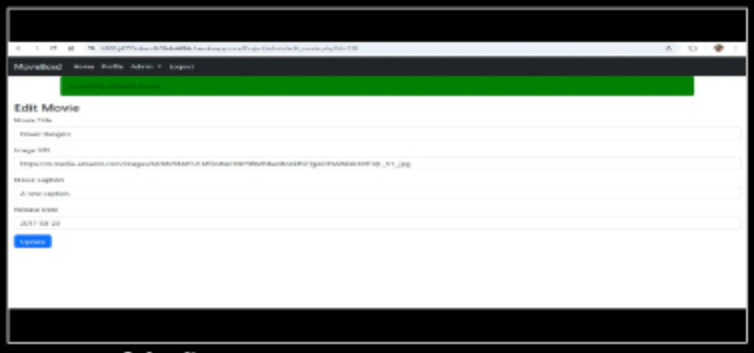
Task #1: Screenshots of the edit page

Sub Task #3: Show an example of successful edit messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1



A sucessful edit

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Edit Data Page
Task #1: Screenshots of the edit page
Sub Task #4: Design/Style should be considered (i.e., bootstrap, custom css, etc)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The edit page

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain your design choices

Response:

The edit page consists of 4 text boxes: one for the title, image url, caption, and release date. The submit button is at the bottom.

End of Task 1

Task 100%

Group: Edit Data Page
Task #2: Screenshots of edit page code
Weight: ~25%
Points: ~0.50

COLLAPSE

Details:
Include ucid/date comments for each code screenshot

Columns: 1

Sub-Task 100%

Group: Edit Data Page
Task #2: Screenshots of edit page code
Sub Task #1: Form should have correct data types for each property being requested

Task Screenshots

Gallery Style: 2 Columns

4 2 1

The form code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the data type choices

Response:

As each type of data being changed is text, all of the forms are text type. Title and release date are required, so I marked those as being requiered.

Sub-Task

100%

Group: Edit Data Page

Task #2: Screenshots of edit page code

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1

JavaScript validation

PHP validation

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the validations

Response:

The validations are similar to the ones from create_movie.php. I check if the title is empty or if it is greater than 200 characters. I check for valid release dates using regular expressions and if the release data isnt empty. Finally, I check if the caption is longer than 500 characters.

Sub-Task

100%

Group: Edit Data Page
Task #2: Screenshots of edit page code
Sub Task #3: Successful edit should have a user-friendly message

Task Screenshots

Gallery Style: 2 Columns

4 2 1

```php
//jd755 11/21/24
if($isValid)
{
    $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("Sucessfully updated movie! ", "success");
    } catch (PDOException $e) {
        movie_check_duplicate($e->errorInfo);
    }
} <- #74-86 if($isValid)
```

Sucessful edit message

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

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

Response:

The ID is grabbed from the URL using $_GET and is combined with an SQL statement to grab the title, image url, caption, and release date of the movie. This data is stored in the $movie variable. Next, the $form variable is created that contains a form for each element to be edited. A for loop is used is change the value of each form to the values in the $movies variable. This populates the forms with the current information. After the user enters new information, this new data is validated. If it passes, an SQL statement is formed that updates the current data. If this is sucessful, a message is sent to the user that everything went well.

Sub-Task

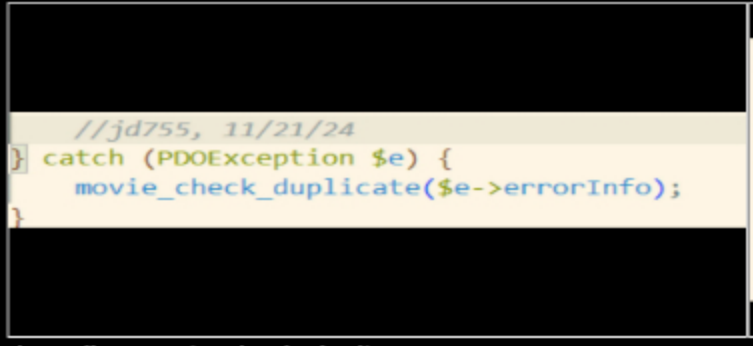
100%

Group: Edit Data Page
Task #2: Screenshots of edit page code
Sub Task #4: Any errors should have user-friendly messages

Task Screenshots

Gallery Style: 2 Columns

4 2 1



the call to movie_check_duplicate

The contents of the function

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the possible errors

Response:

As movies are uniquely identified by both their title and release data, if a user edits a movie with matching information as another, this will throw an error.

Sub-Task

100%

Group: Edit Data Page

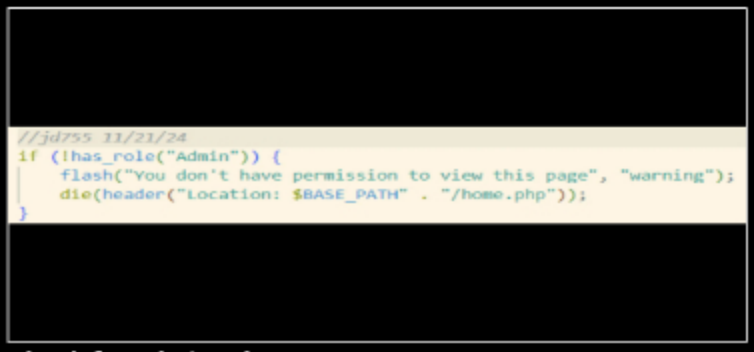
Task #2: Screenshots of edit page code

Sub Task #5: Include any other rules like role guards and login checks

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Check for admin role

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Briefly explain the logic/reasoning

Response:

Non admin users have no control over the contents of the movies themselves. The data can only be from actual verified information from the API or from trusted sources (the admins).

End of Task 2

Task

100%

Group: Edit Data Page
Task #3: Screenshot of records from DB
Weight: ~25%
Points: ~0.50

COLLAPSE

Sub-Task

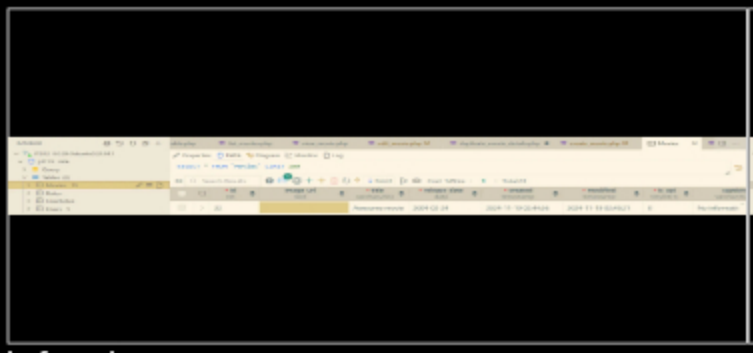
100%

Group: Edit Data Page
Task #3: Screenshot of records from DB
Sub Task #1: Show a before and after screenshot of the record (two screenshots)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



before change

After change

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain what differs

Response:

I changed the image_url section

End of Task 3

Task

100%

Group: Edit Data Page
Task #4: Add related links
Weight: ~25%
Points: ~0.50

COLLAPSE

Checklist

*The checkboxes are for your own tracking

#	Details

#1	Include the heroku prod link for this page
#2	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

Task URLs

URL #1
https://it202-jd755-prod-028c8689138c.herokuapp.com/Project/admin/edit_movie.php?id=130

URL
https://it202-jd755-prod-028c8689138c.herokuap

URL #2
https://github.com/legitdaylight/jd755-IT202-007/pull/50

URL
https://github.com/legitdaylight/jd755-IT202-007

End of Task 4

End of Group: Edit Data Page
Task Status: 4/4

Group

100%

Group: Delete Handling
Tasks: 3
Points: 1

COLLAPSE

Task

100%

Group: Delete Handling
Task #1: Screenshots related to delete
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:
Heroku dev url must be visible in all relevant screenshots

Include ucid/date comments for each code screenshot

Columns: 1

Sub-Task

100%

Group: Delete Handling
Task #1: Screenshots related to delete
Sub Task #1: Show the success message of a delete

Task Screenshots

Gallery Style: 2 Columns

4 2 1

successful delete

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Sub-Task 100%

Group: Delete Handling

Task #1: Screenshots related to delete

Sub Task #2: Show any error messages of a failed delete (like id not being passed)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Invalid ID

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

When the button to delete is clicked, the ID is sent to delete_movie.php. There, it checks if the ID is valid. A valid is ID is one that is greater than -1. If the ID is not valid, it gives the error message and redirects back to the list page.

Sub-Task 100%

Group: Delete Handling

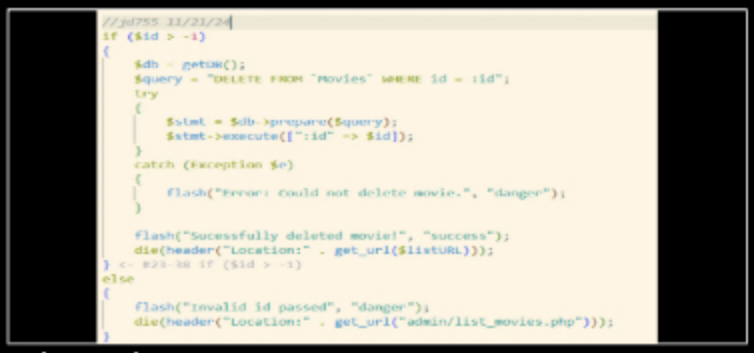
Task #1: Screenshots related to delete

Sub Task #3: Show the code related to the delete processing

Task Screenshots

Gallery Style: 2 Columns

4 2 1



```php
//jd755 11/21/24
if ($id > -1)
{
    $db = getDB();
    $query = "DELETE FROM `Movies` WHERE id = :id";
    try
    {
        $stmt = $db->prepare($query);
        $stmt->execute([":id" => $id]);
    }
    catch (Exception $e)
    {
        flash("Error: could not delete movie.", "danger");
    }

    flash("Sucessfully deleted movie!", "success");
    die(header("Location:" . get_url($listURL)));
} <-- #23-38 if ($id > -1)
else
{
    flash("invalid id passed", "danger");
    die(header("Location:" . get_url("admin/list_movies.php")));
}
```

Delete code

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Explain in concise steps how this logically works

Response:

If the ID is valid, an SQL query is created that removes the item in the database that matches with the entered ID. If this is successful, a message is displayed and the user is redirected back to the list page with the same filter parameters.

Sub-Task

100%

Group: Delete Handling
Task #1: Screenshots related to delete
Sub Task #4: Explain the delete logic

Task Response Prompt

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

Response:

This is a hard delete as there is not point in keeping invalid data in the database. Sometimes the API returns movies that aren't very useful for the user. Only the admins can delete data.

End of Task 1

Task

100%

Group: Delete Handling
Task #2: Screenshots of the data
Weight: ~33%
Points: ~0.33

COLLAPSE

Sub-Task

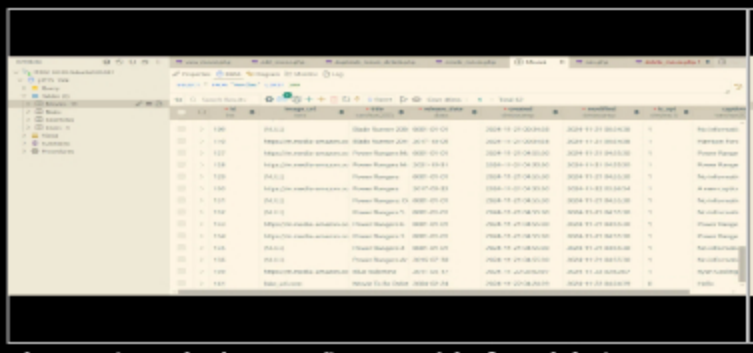
100%

Group: Delete Handling
Task #2: Screenshots of the data
Sub Task #1: Show a before and after screenshot of the DB data (two screenshots)

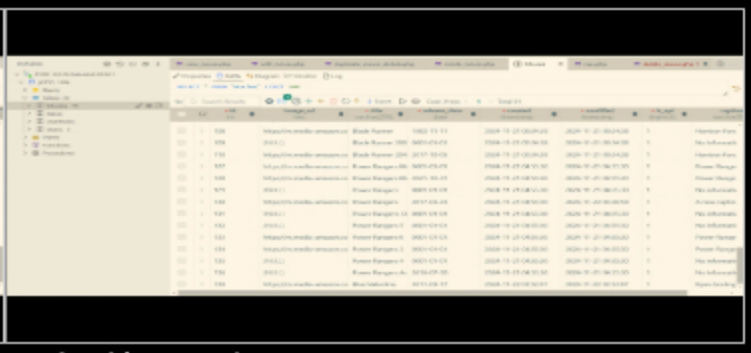
Task Screenshots

Gallery Style: 2 Columns

4 2 1



The movie at the bottom (item 141) before deletion



Movie with ID 141 is now gone.

Caption(s) (required) ✓

Caption Hint: *Describe/highlight what's being shown (note precisely what changed)*

End of Task 2

Task

100%

Group: Delete Handling
Task #3: Add the pull request link for the branch related to this feature
Weight: ~33%
Points: ~0.33

COLLAPSE

Details:

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

Task URLs

URL #1

https://github.com/legitdaylight/jd755-IT202-007/pull/55

URL

https://github.com/legitdaylight/jd755-IT202-007

End of Task 3

End of Group: Delete Handling
Task Status: 3/3

Group

100%

Group: Misc
Tasks: 4
Points: 1

COLLAPSE

Task

100%

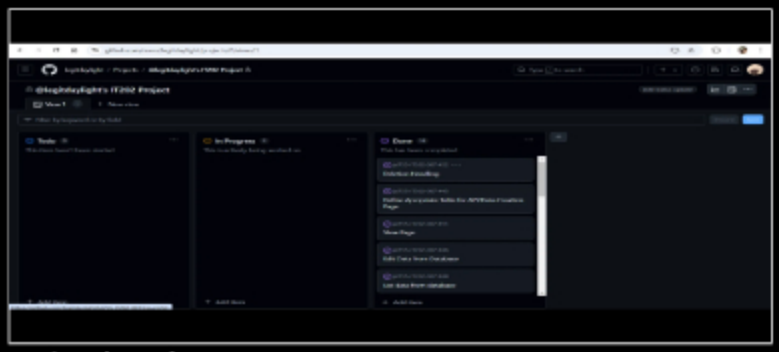
Group: Misc
Task #1: Screenshot of your project board from GitHub (tasks should be in the proper column)
Weight: ~25%
Points: ~0.25

COLLAPSE

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Project board

End of Task 1

Task

100%

Group: Misc
Task #2: Provide a direct link to the project board on GitHub
Weight: ~25%
Points: ~0.25

COLLAPSE

Task URLs

URL #1
https://github.com/users/legitdaylight/projects/2/views/1

URL
https://github.com/users/legitdaylight/projects/2

End of Task 2

Task

100%

Group: Misc
Task #3: Talk about any issues or learnings during this assignment
Weight: ~25%
Points: ~0.25

COLLAPSE

Task Response Prompt

Response:

I faced quite of bit of trouble during this assignment. There were a lot of challenges for me to overcome to make this project work. The main two challenges was having the database accept bulk data from the API and to maintain the filtering parameters after deleting a movie. Those two alone took up too much time to figure out.

End of Task 3

Task

100%

Group: Misc
Task #4: WakaTime Screenshot
Weight: ~25%
Points: ~0.25

COLLAPSE

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: 2 Columns

4 2 1

Wakatime screenshot 1

WakaTime screenshot 2

End of Task 4

End of Group: Misc
Task Status: 4/4

End of Assignment