Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-007-F2024/it202-api-project-milestone-2-2024-m24/grade/ob75

Course: IT202-007-F2024
Assigment: [IT202] API Project Milestone 2 2024 M24
Student: Omar B. (ob75)

Submissions:

Submission Selection

1 Submission [submitted] 12/9/2024 7:03:22 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

100%

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

COLLAPSE

Task

100%

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

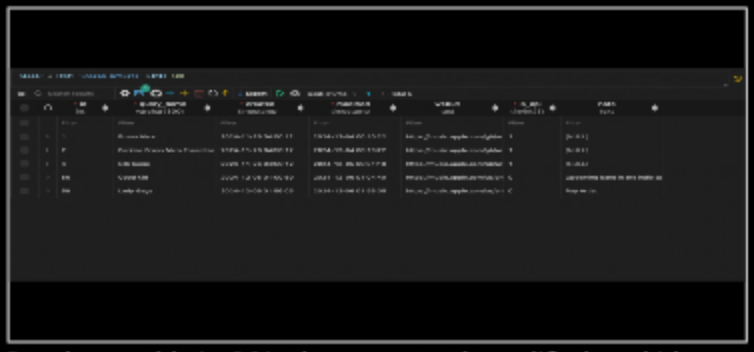
100%

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



Database table in SQL shows created, modified, and id.
Also contains addition appropriate columns.

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:

There is a column for id to represent each artist's ID in the database for when they are fetched or created, the query_name column to represent each artist's name, created column to represent when that artist was inserted/created, modified column to show when that artist is edited or modified, the weburl column to represent the url to the artist's music profile (mainly has been apple music profiles), is_api column to show if the artist was taken from API data(1) or created(0), and lastly the note column that represents any information the user would insert in for the artist.

End of Task 1

Task

Group: Define Appropriate Tables for Data
Task #2: Add the pull request link for the branch related to this feature

100%

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/OmarBarrera1/ob75-IT202-007/pull/44

https://github.com/OmarBarrera1/ob75-IT202-007

End of Task 2

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

Group

96%

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

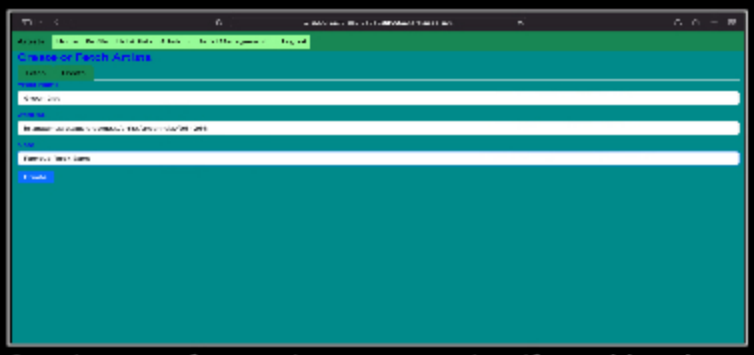
100%

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



Creation page for creating custom artists if not able to be fetched.

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

fetch_artists function created on line 3 with UCID on top to fetch artists from API.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the code

Response:

The code here created a function named fetch_artist that takes in the artName variable. The function associated the artName variable for what artist name the admin is entering in for the fetch to be taken from the API in its parameter that it uses which is query on the API. The endpoint for the API data is used and stored and also tells what API is being used to fetch from. The result is then stored with the array of data from the API and is then used to extract specific data from the API. An if statement is then done to extract the data from the API and specify what data I want (in this case, artist name and their web url) by looping through the array looking for when there is the instance of ['artists']['name'] and ['artists']['weburl'] and setting them to the specified columns in my database. The artist variable

that now contains the new extracted data from the api is then set to result variable to update it and return that value.

Sub-Task

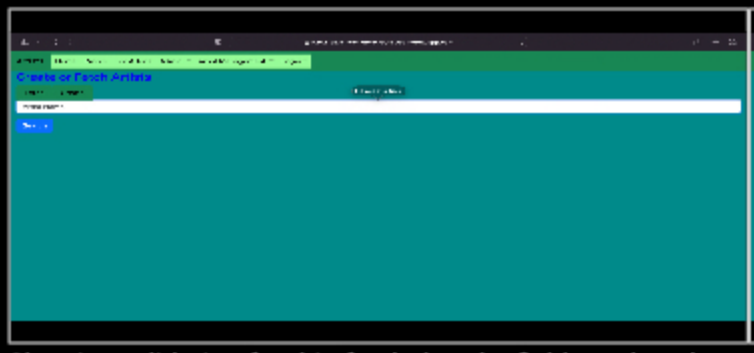
100%

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

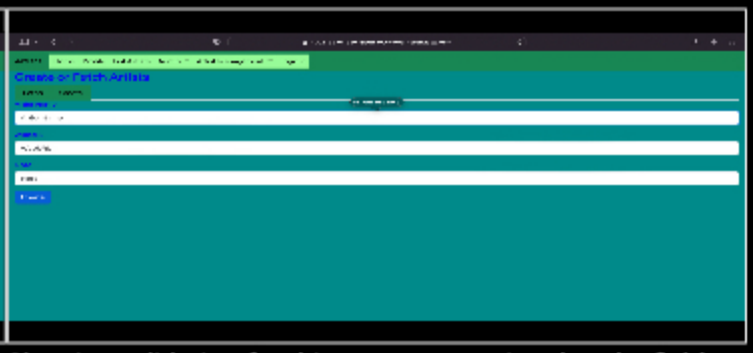
Task Screenshots

Gallery Style: 2 Columns

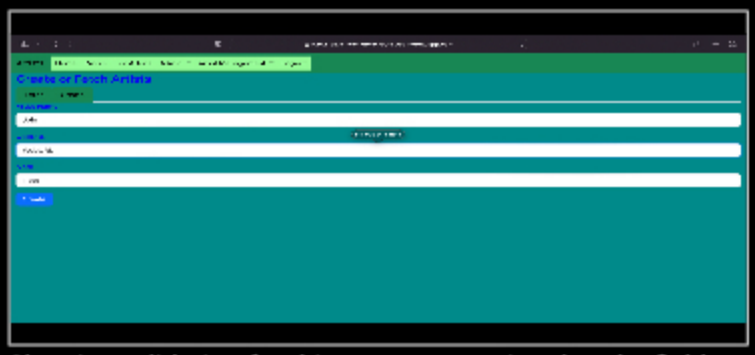
4 2 1



Showing validation for this fetch that the field needs to be filled out for Artist Name.



Showing validation for this custom creation that the field needs to be filled out for Artist Name.



Showing validation for this custom creation that the field needs to be filled out for webURL.

Caption(s) (required) ✓

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

Sub-Task

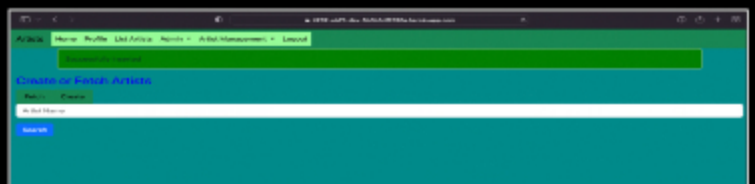
100%

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



Shows successful insertion after fetching or custom creating an artist.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Data Creation Page

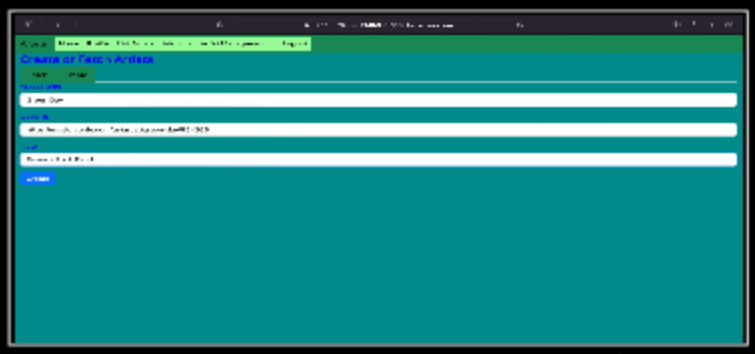
Task #1: Screenshots of the creation page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Bootstrap implemented to change design/style of page.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain your design choices

Response:

The bootstrap design allowed for a much cleaner view of the menu and page and allows for the admin to easily insert data into the correct fields and be able to stop clutter from confusing the admin.

End of Task 1

Task

85%

Group: Data Creation Page

Task #2: Screenshots of creation page code

Weight: ~25%

Points: ~0.50

COLLAPSE

Details:

Details:

Include ucid/date comments for each code screenshot

Columns: 1

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

Screenshot Showing form for creation page and each type being requested.

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 the form on the fetch side that has a type field for fetch that is hidden for when the admin wants to fetch the data, the field type requested was for a text input and would be named 'artists' where when the user wants to fetch data, they would just put the artist name and would be able to search(that works as a submit to the data) which would get the data. The create side uses the same concept but changes its hidden field type to create for when the admin wants to create instead, and will require still the artists name but also additional field types of text for Web Url and Note that will be required for the user to insert those data in to be able to create their artist. It also contains the submit type field that is named create.

Sub-Task

0%

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

Missing Caption

Caption(s) (required)

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

Missing caption(s)

Task Response Prompt

Briefly explain the validations

Response:

Missing Response

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

Screenshot of code showing successful insertion. Relevant lines from 54 - 58 for user friendly message.

Caption(s) (required) ✓

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

Task Response Prompt

Explain how duplicate/existing data is handled

Response:

Duplicate and existing is handled by the line of code at line 52 that inserts the record into the database `Shazam-Artists` with the artist data and setting the update_duplicate to true to allow for any existing records that match the same data that was inserted to just be updated instead of just having another instance of that record.

Sub-Task

Group: Data Creation Page

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

Screenshot showing user-friendly messages for errors from line 54 - 73 for any of the conditions.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly describe the scenarios

Response:

Some scenarios in which there would be an error would be in cases where invalid information is fetched from the API data such as inserting numbers into artist name field instead of alphabetic characters which would lead it into not finding a valid record to insert and will flash the user that Invalid data was passed. Another would be through any database errors where the database is not able to be reached and could cause a flash of "database error". Lastly "invalid data records" flash could be caused when certain data does not match the data type in the database table.

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

Showing the process for fetching data if user fetches, Line 14 - 29 shows fetch process.

Screenshot showing the fetch_artists function being called and fetching the data and mapping.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the steps (include how duplicates are handled)

Response:

When fetching for API data, a $_Post action is done to get what field type(button on page) the user decided to click on and submit. An empty array is created for artist to take in the data later on. An if statement is now done to check when the user clicks on fetch and will take the artists name that the user enters in named artName. If that artname is entered, then the fetch_artist function is called to go through the process of extracting and getting the data from the specified api. If that artist was able to be fetched, then array_map is done to set each value in the array to the previously created value named artists, set them with the default value of one with is_api for every value to demonstrate that its from the api and return each value. It is then inserted after into the database with duplicates being handled when the data is being inserted by setting 'update_duplicate' to true so that existing records of the same value are rather updated.

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

Screenshot showing setting each artists[is_api] to 1 to distinguish between custom and API data. Line 25 specifically.

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:

If the fetch action is done and the admin is fetching the data, then the data that would be mapped to the database would also be included with the is_api(which defaults to 1) and be marked with it to show what data has been fetched from the API side. Any that don't inlcude is_api shows that it was created manually and not fetched from the

API data.

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

Screenshot showing role guard for admin only access. Line 5 - 8 shows relevant code.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic/reasoning

Response:

The Role guard is set at the top of the code to first check whether that user has the admin role in order to access the page and if not, it would give them a user-friendly message and take them back to the home page. If they do have the admin role, then they will be able to access the fetch and create page. This is done to prevent regular users from adding random data or extra data to the database whereas it should only be special privileged users that should take care of adding to the database.

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

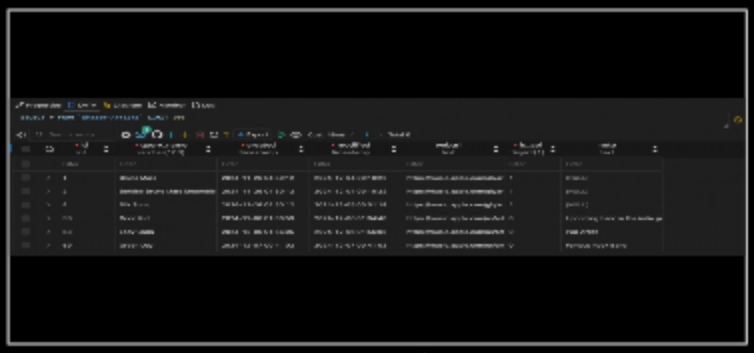
100%

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



Showing database table with the fetched API data being the data marked 1 in the is_api column.

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

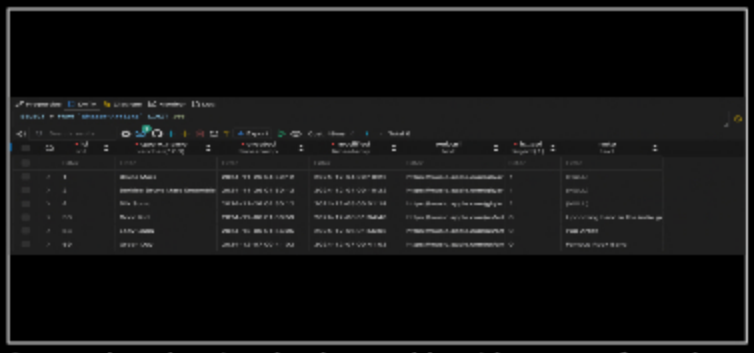
Task #3: Screenshot of records from DB

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing database table with custom form data being entered, is_api is marked 0 for custom created data under column.

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://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/project/admin/create_artist.php

https://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/project/admin/create_artist.php

URL #2 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/47

End of Task 4

End of Group: Data Creation Page
Task Status: 3/4

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

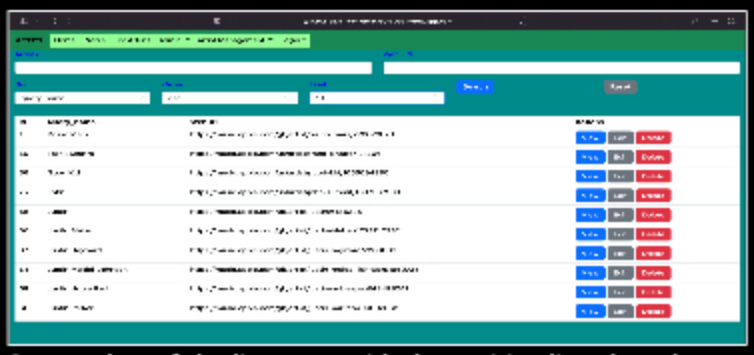
100%

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



Screenshot of the list page with the entities listed on the page.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly describe the page

Response:

The data list page contains a first few information about each record in the database and displays it to the viewer. The id, query_name(artist name), and weburl are displayed along side the edit, view, and delete option buttons beside the to use those funcitons. There are also fields to type text into for example Artists and WebURL to filter the search result and also sort by either query_name or weburl. There is also another field for order to sort from asc or desc order of the records filtered. Lastly there is a limit field that allows the limit on how many records should appear on the list.

Sub-Task

100%

Group: Data List Page (many entities)

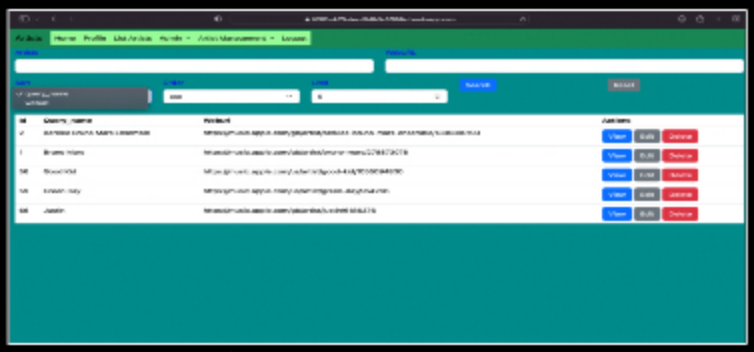
Task #1: Screenshots of the list page

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



Showing the sort form on page and also showing the limit

set to 5 and showing 5 records only. (defaults the limit to 10).

Caption(s) (required) ✓

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

Task Response Prompt

Briefly mention what's available to the user

Response:

The id, query_name (artist name), and web URL are given with buttons for editing, viewing, and deleting, with fields to filter search results by artist or web URL, sort by either query_name or web URL, choose ascending or descending order, and set a limit on the number of records displayed.

Sub-Task

100%

Group: Data List Page (many entities)

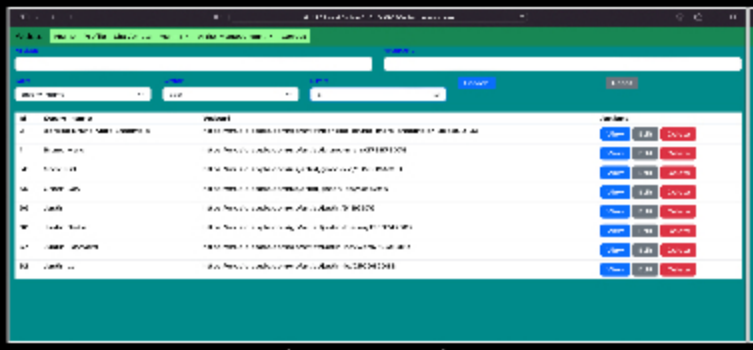
Task #1: Screenshots of the list page

Sub Task #3: Demonstrate a few varied filters/sorts

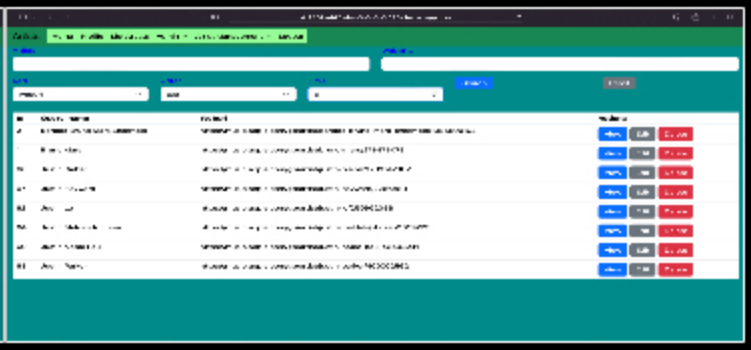
Task Screenshots

Gallery Style: 2 Columns

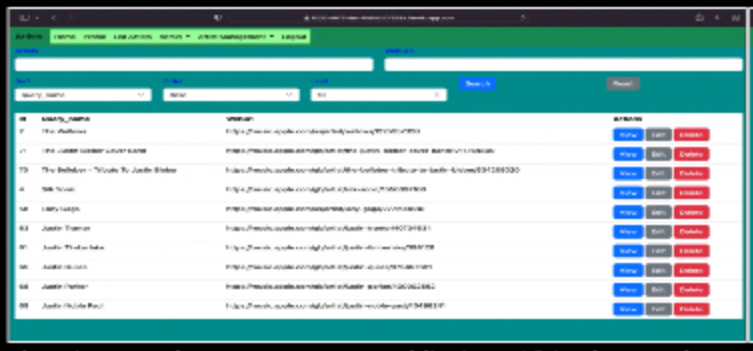
4 2 1



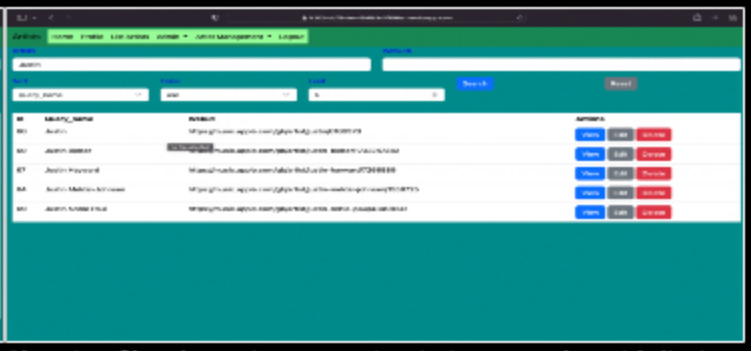
showing sort by name(query_name) and limit to 8 in asc order.



showing sort by web url and limit to 8 in asc order.



showing sort by query_name and limit to 10 in desc order.



Showing filter by artist name Justin in asc order with limit of 5

Showing filter by weburl name and only one record perfectly matching that.

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

Screenshot showing when there isn't a record and shows "no records to show" as the message.

Caption(s) (required) ✓

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

Sub-Task

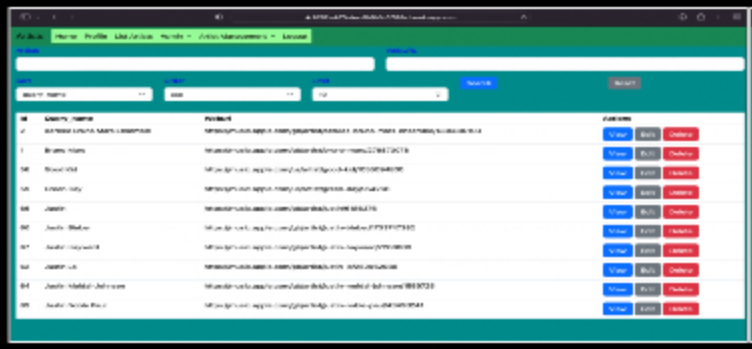
100%

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)

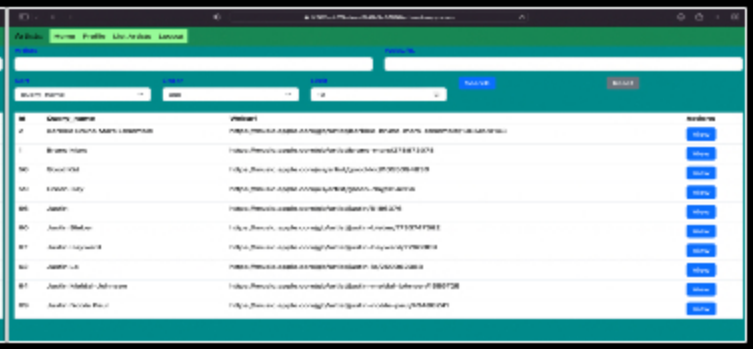
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing admin side of list page with link to view, edit and delete.



Screenshot showing regular user list page. Only view link is enabled for regular users without admin role.

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:

The only users that are able to edit and delete links are the users that have the admin role to stop regular users from deleting the records or editing the records from the database. However, the users that can interact with the view links is everyone since it is fine to allow anyone to check the information that is given on each record.

Sub-Task

100%

Group: Data List Page (many entities)

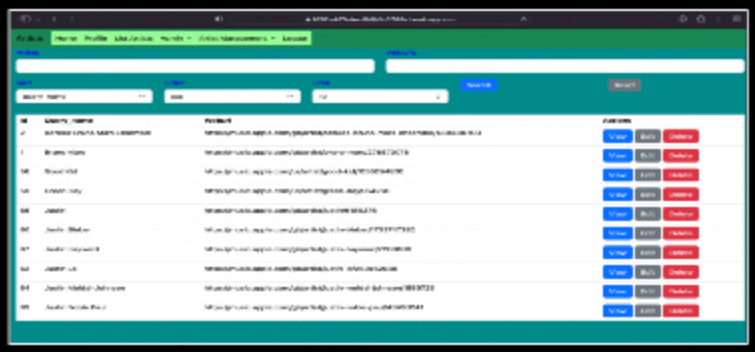
Task #1: Screenshots of the list page

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



Screenshot of entity showing summary which is the id, query_name(artist name) and Weburl.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Data List Page (many entities)

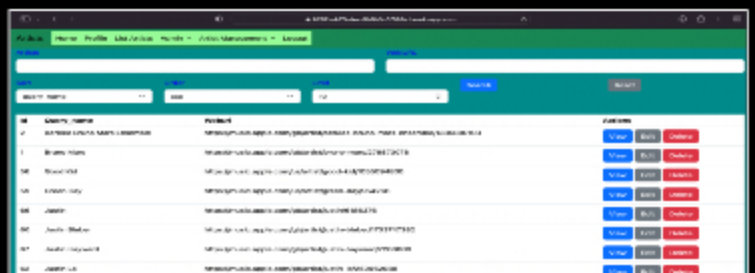
Task #1: Screenshots of the list page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing bootstrap applied to the list page with buttons and fields and sorted nav links at top.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain your design choices

Response:

The Bootstrap design along with the custom css allowed for a much clearer view on the information on each other and also allowed for the buttons to appear on the side of each artist so it becomes accessible to click on without the need of creating more code to style it.

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

Screenshot showing sort form generation being Search and including all other filters and part of line 113.

Screenshot showing sort form generation and rest of line 113.

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:

The form generation here for sorting is done first with creating a row and setting field types for the artist name and/or web url so that it can filtered when the users types in specific entities into those fields to filter from. Then another row is created for the rest of the sorts to filter from such as filtering by query_name first or filtering by weburl first by using a select box to select either of the two. The next sort is ascending order or descending order that also uses a select box so that the user selects which to sort from. The limit filter is done by setting a default type -> number to 10 and can be dynamically changed after by changing the value to set the number of records that should appear. The submit box is lastly created named as search to search for when the filters are applied and will submit the filters to output the new form generation.

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

Screenshot showing the db query line 29 and line 50-54 and sort being handled line 55-70.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the related logic

Response:

In here firstly we call the database using SELECT and specify the fields that columns of the table that are going to be used and from the `Shazam-Artists` table that is then stored into the sql variable. After, if functions are created for both artists and weburl to check for similar mathces to the text that the user would type in to the filter fields for each respective field. The limit function is also done here and set to a default of 10 if not specified or if it is less than 0 or greater than 100, and will also allow for the limit to be chaged to different values in that range and will send it over

the filter. Then the sql data extracted from the table is ordered by asc or desc order (whichever the user specifies) and will now apply all the filters that are set to the field types for artists name, web url, and/or limit. It will then display that new data to the user.

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

Screenshot showing how the data is populated and displayed onto the web page line 76-87.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the related logic

Response:

Ignore columns is made here to ignore the created, modified, is_api, and id(note that id is still shown to the user to properly get the id of user) from being shown to the user in the list page. The table is then generated to take in the columns to display to the user and the new results from the filter previously processed is then set to 'data' in the table to generate the new data. The ignore columns is also set here to ignore the set columns called and is set to 'ignore columns' the view link is also set in the table for all types of user roles to view the artist in the list page. The edit and delete links are only available to show on the table if the user has the admin role.

Sub-Task

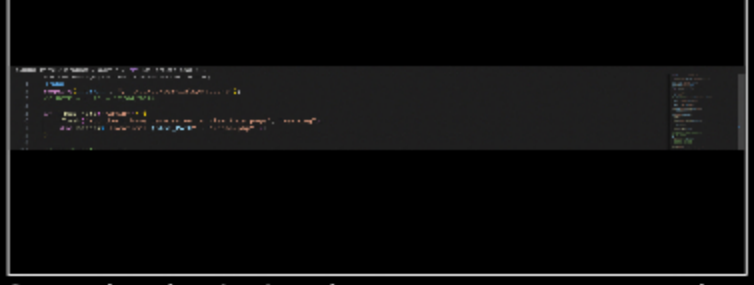
100%

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



Screenshot showing in order to access page, user must be admin and shows appropriate if not. line 5-8

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic/reasoning

Response:

An if statement is done at the top of this page to check when the user does not have the admin role and will send them back to the home page if they do not have the role and is done to differentiate 2 separate list pages for the admin and the user. The one for admin would allow them to have the whole list page along with all 3 links for viewing, editing, and deleting. The list page for regular users would only have the list page with the view link.

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-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

https://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

URL #2 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/48

URL #3 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/49

URL #4 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/50

URL #5
https://github.com/OmarBarrera1/ob75-IT202-007/pull/53

https://github.com/OmarBarrera1/ob75-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

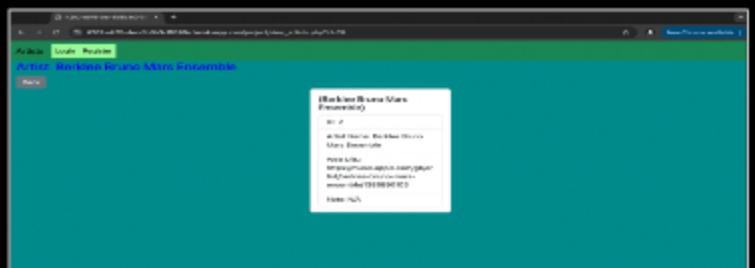
100%

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



Screenshot showing entity is fetched by id via the url at the top.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: View Details Page (single entity)

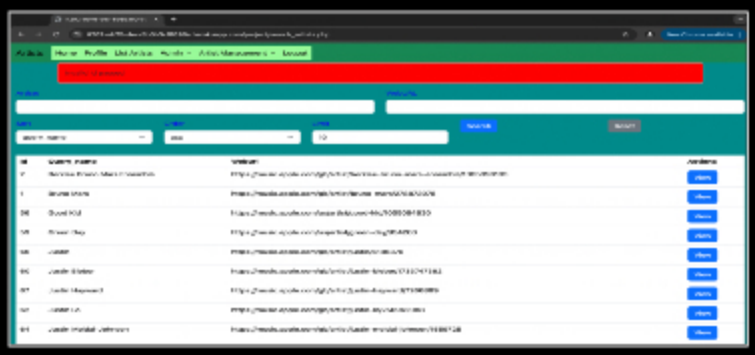
Task #1: Screenshots of the details page

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



Screenshot showing when missing id is applied, user is redirected to list page with "Invalid id passed" as the message.

Caption(s) (required) ✓

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

Sub-Task

100%

Group: View Details Page (single entity)

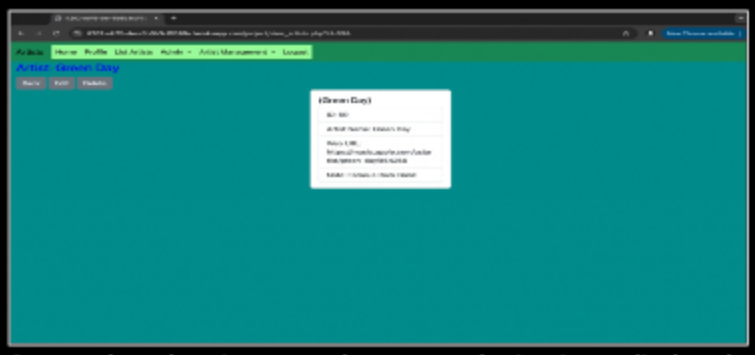
Task #1: Screenshots of the details page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing same bootstrap design is applied and showing details of artist.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain your design choices

Response:

The design here allows for a single view of each card's detailed information being the note added during custom creation of that artist or editing that arist. The design also allows for back button, edit button, and delete button links to be accessed directly from the view page without having to return back to list page to do it.

Sub-Task

100%

Group: View Details Page (single entity)

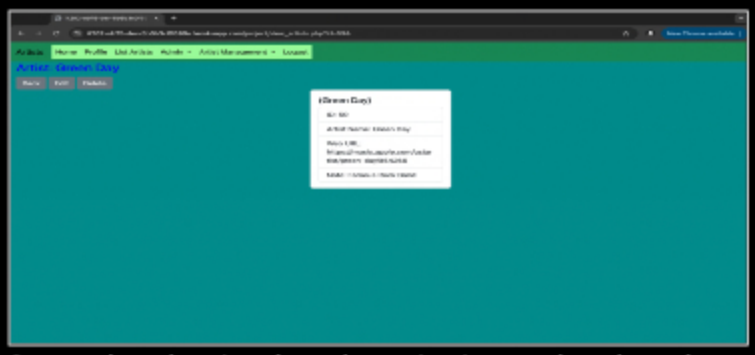
Task #1: Screenshots of the details page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing data shown in view to also show the notes placed by the admin editing or creating the artists.

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 difference in the view page compared to the list page is that the view page now contains any notes that were added about the artist during the custom creation of that artist or when the user decided to edit the artist and add a note. The note can be any description about the artist and give more details than the list page would give on the artist, in which the list page only shows the id, name, and web url.

Sub-Task

100%

Group: View Details Page (single entity)

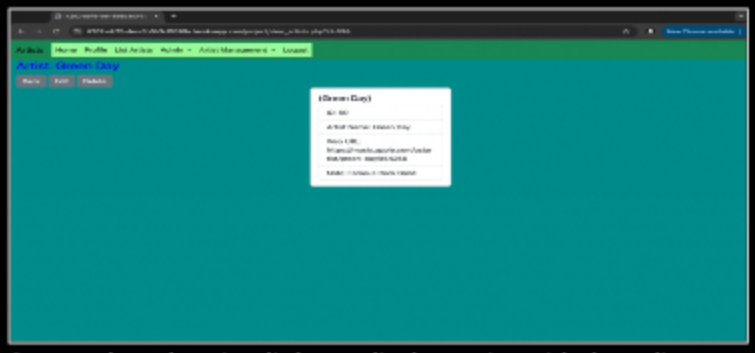
Task #1: Screenshots of the details page

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



Screenshot showing link to edit the entity with the edit button on the left(right beside the back button and only for admin).

Caption(s) (required) ✓

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

Sub-Task

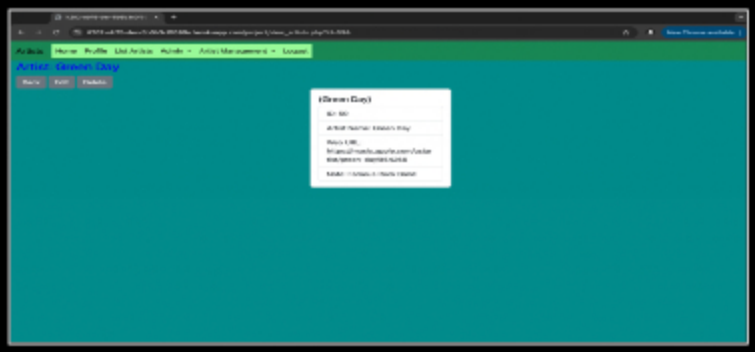
100%

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



Screenshot showing link to delete the entity with the delete button on the left (right beside the edit button & only for admin).

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

Screenshot showing id being fetched on line 8.

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 logic here is that the id is obtained anytime the view button link is clicked on any specific artist/entity and is then ran through the entire list in the database until the id is found that matches the one clicked on with the view button link. If an id is not specified or is one that does not exist in the database, then the try catch block will catch that instance and will provide the user-friendly message that there was an error fetching the record. If the id number is not greater than -1 however, then there will be a different message saying "invalid id passed" and will flash that to the user.

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

Screenshot showing the DB query to get record on line 16.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

The DB query here is getting all the columns in the database from the `Shazam-Artists` table using the WHERE clause to match where the id mathces the id of the artist that is going to be viewed. The artist is then prepared to be viewed and displays the information to the user.

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

Screenshot showing form that presents data and shows links such as back button, edit, and delete when viewing.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic

Response:

When the view button link is clicked then we show the user what artists is being viewed. Then an if statement is created to for when the user has the admin role, it will provide them with 3 button links; back that would take them

back to the list page, edit which would take them to edit the artist, and delete which would delete the artist record. If the user does not have the admin role, then they are only provided with the back link to take them back to the regular user list page. The view page is populated with first the artist's id in the database, then their name, then the weburl, and lastly the note provided by the user. This is all created on a card that is centered in the screen to display much better to the user.

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-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

URL #2 https://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/search_artists.php

URL #3 https://github.com/OmarBarrera1/ob75-IT202-007/pull/51

End of Task 3

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

Group

95%

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

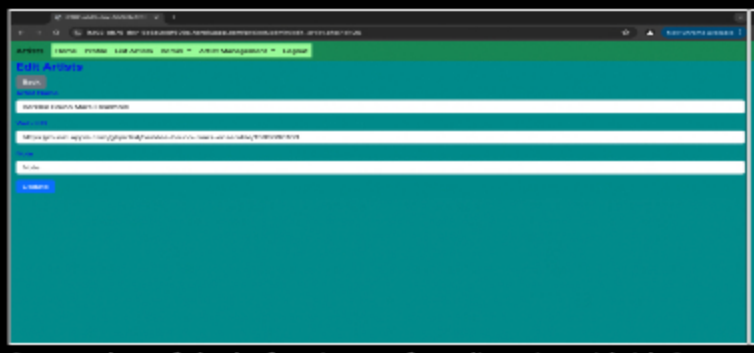
100%

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)

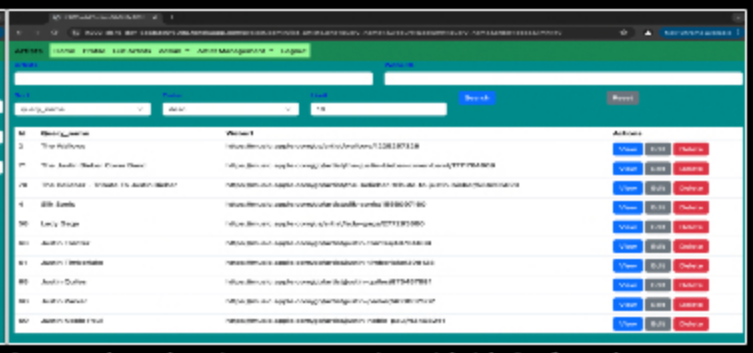
Task Screenshots

Gallery Style: 2 Columns

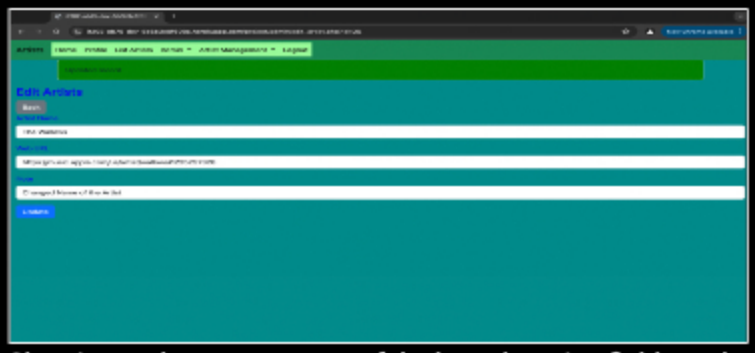
4 2 1



Screenshot of the before image for edit artist with id=2.



Screenshot showing same artist with id=2 after change image with new updated name and url.



Showing update was successful when changing fields and update.

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

Task Response Prompt

Briefly explain what you changed

Response:

Response:

The artist name in the screenshot previously was named Berklee Bruno Mars Ensemble and the web url for that name, and no note was provided, with the id of 2 from the database. The edit button is clicked and then the artists name is changed to The Wallows and changed the web url to correspond with the new artist name for The Wallows and also edited the note to include a brief description saying "changed name of Artist" to show change. The id is still the same for the record (id = 2) but the rest of the information data has changed.

Sub-Task

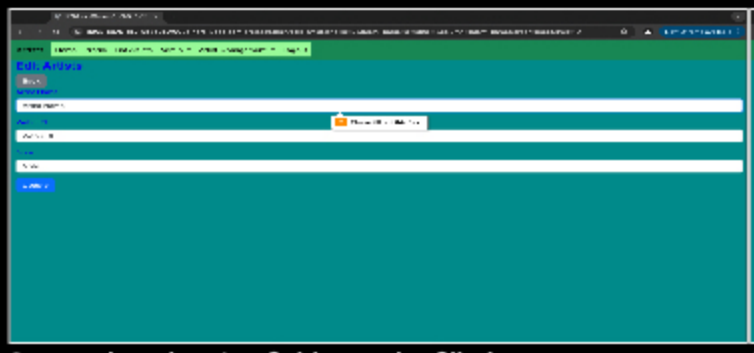
100%

Group: Edit Data Page
Task #1: Screenshots of the edit page
Sub Task #2: Show examples of validation messages

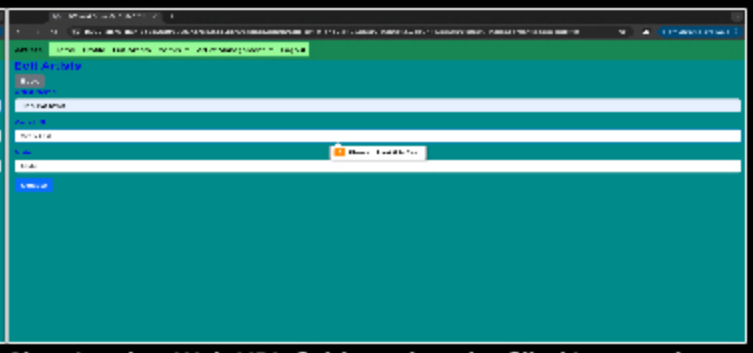
Task Screenshots

Gallery Style: 2 Columns

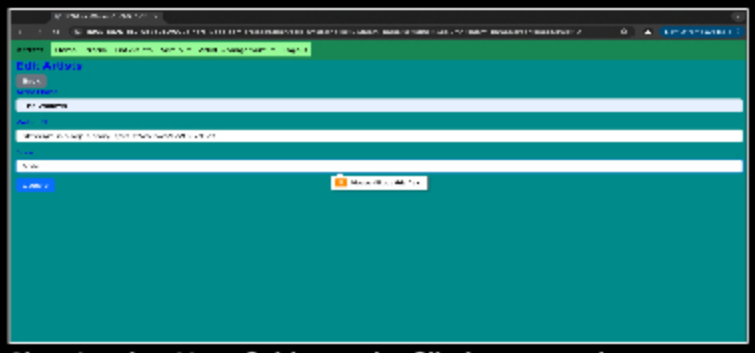
4 2 1



Screenshot showing field must be filled out



Showing that Web URL field needs to be filled in to update.



Showing that Note field must be filled out to update.

Caption(s) (required) ✓

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

Sub-Task

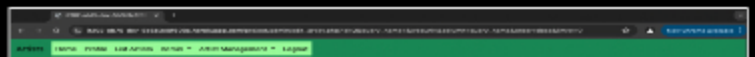
100%

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



Showing success edit by showing message "Updated record"

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Edit Data Page

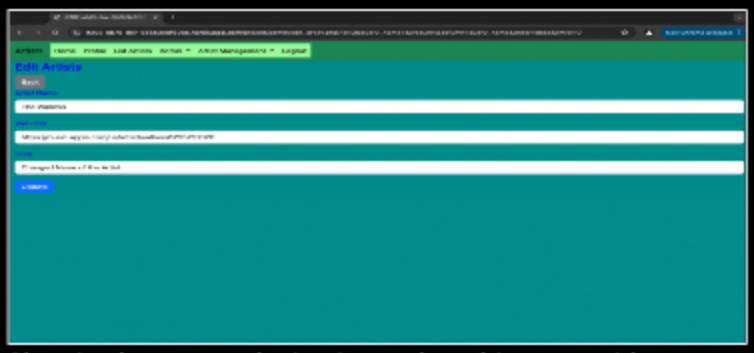
Task #1: Screenshots of the edit page

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Showing bootstrap design is used on this page with custom CSS.

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain your design choices

Response:

The design here allows for text boxes to be inserted and allows for the user to edit the text boxes that are already preloaded with the artist's information so that the user knows what entity they are editing. It also allows for an easy button link for the back button to go back to the list page and a update button that would update the record when changing the information.

End of Task 1

Task

80%

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

Showing form types being requested & required for the user to fill out to update. Line 74-87 and also line 96-100 for form.

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

Task Response Prompt

Briefly explain the data type choices
Response:

The data type choices here were made for first the input field for query_name(artist name), the weburl, and note which are all required to have data inputted inside those fields to be able to update the record. All these data types are necessary as they are what represent the main data in the database and will need some sort of information in them to be created or updated. The back button is also created to take them back to the list page using the link for admin and update button link is also done to update the record for when the user has filled out the necessary information.

Sub-Task
0%
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

Missing Caption

Caption(s) (required)

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

Missing caption(s)

Task Response Prompt

Briefly explain the validations

Response:

Missing Response

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

Showing successful message on line 46 if edit was successful.

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 of the artist that is being edited gets stored into the id variable, then the if statement is done for when the artist name is being changed and will set the new data name to artName. We then open the database and set the update of the database table `Shazam-Artists` to the variable query. Then we append to the query for each of the fields that is being updated for the artist and append the id of that artist as well since we will keep the id the same. Then we execute the query that contains the updated records and will flash a message to the user "Update record" and if not able to update record, it will flash a different message "An error occurred". First with fetching, we set artist to an empty array and attempt to obtain the database record when the id is greater than -1. We then grab the data id, query_name, weburl, note from `Shazam-Artists` where our artist id that we are editing mathes the id in the database. We then fetched our record data from the database and store it into artist variable and if any error occurs, we appriopriately show the user-friendly messages to the user that there was an error. If that artist is being edited than we then set all the key in our record that was fetched to keys variable. We then populate the form with the existing data record that is stored in the database and will then be able to be edited by the user to change and update the specific fields.

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

Showing error message on line 49 if an error occurs.

Showing flash message(line 70) if invalid id is passed or error fetching record(line 67).

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the possible errors

Response:

An instance where a possible error for line 49 can occur would be when invalid data is being put into the input fields to be updated and will show the user that "an error occured". For line 70, the situation would be when an invalid id is passed into the edit and will not be able to edit a non existing record so a flash of "invalid id passed" would appear. And lastly for line 67 would occur when there is an error fetching the record from the database and would appear a message to the user saying "Error fetching record".

Sub-Task

Group: Edit Data Page

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

Screenshot showing user must be admin to access page line 5-8 and UCID underneath

Caption(s) (required) ✓

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

Task Response Prompt

Briefly explain the logic/reasoning

Response:

The role guard here is created to stop an non admin role users from being able to access the edit function and editing or updating any of the records in the database since we only want the admin role to be able to do that. If they do not have the admin role, then they are sent back to the home page of their user profile.

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%

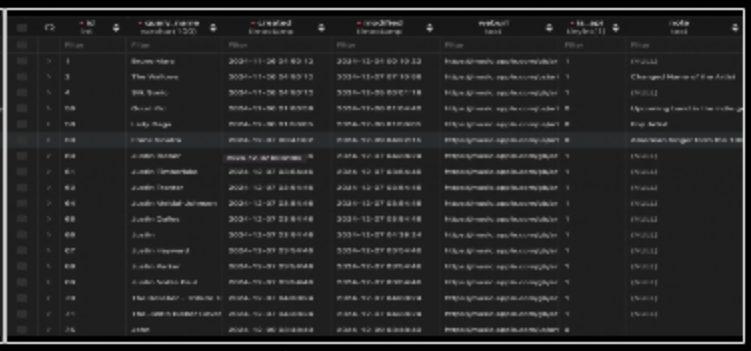
Group: Edit Data Page
Task #3: Screenshot of records from DB
Sub Task #1: Show a before and after screenshot of the record (two screenshots)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

Screenshot showing before edit is done in database. Will do edit on ID 59(Green Day).



Screenshot showing after edit is done and in database new name, web url and note was change for ID 59 (Frank Sinatra now.)

Caption(s) (required) ✓

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

Task Response Prompt

Explain what differs

Response:

In the first screenshot showing the before edit feature is done, the id of 59 with the query_name Green Day is in the database with the respective web url and the note created for them (says "Famous Rock Band"). When the edit is done on this id 59 entity, then the query_name, weburl, and note changes to be Frank Sinatra for the query_name, its respective weburl, and a different note now that says "American singer from the 1900s". The modified column has also changed to be from the timestamp that it was edited compared to what it was before.

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-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

URL #2 https://github.com/OmarBarrera1/ob75-IT202-007/pull/50

End of Task 4

End of Group: Edit Data Page
Task Status: 3/4

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

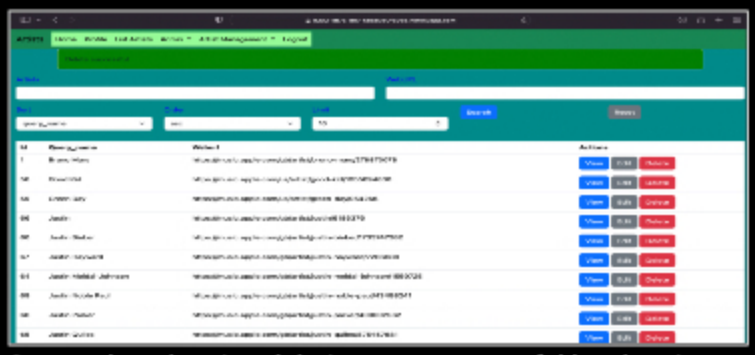
100%

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



Screenshot showing deletion was successful by showing message "Delete successful".

Caption(s) (required) ✓

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

Sub-Task

100%

Group: Delete Handling

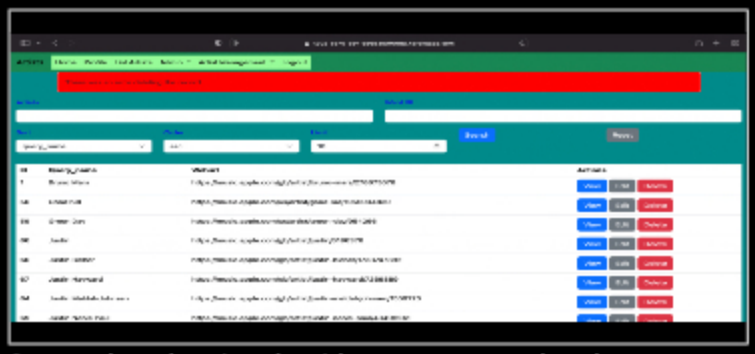
Task #1: Screenshots related to delete

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



Screenshot showing that id was not passed and error occurred when trying to delete the record.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

When deleting a record, there is an id associated with each record that the delete link will fetch for when clicked on. When that delete action is done, the id is fetched on which artist the user is trying to delete and will try to match it against the database, seeing for which id in the database matches the id fetched. When there is no id given or an incorrect id is placed, then the delete action will not be able to fetch any data since it does not exist and will provide the user with a flash saying there was an error deleting the record.

Sub-Task

100%

Group: Delete Handling

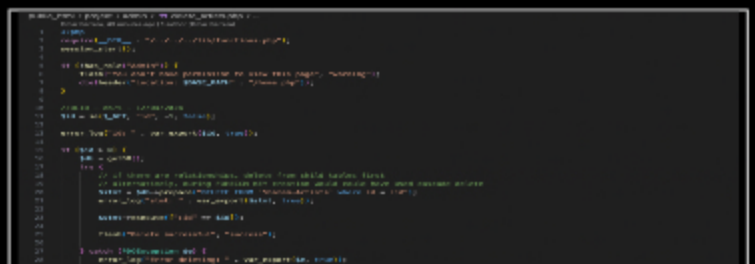
Task #1: Screenshots related to delete

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

Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing delete processing for record. Line 15 - 33 and line 20 showing actual delete from the database.

Caption(s) (required) ✓

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

Task Response Prompt

Explain in concise steps how this logically works

Response:

Firstly, the id of the artist is fetched of the delete action that was done and will store that into the id variable. Then an if statement is created to check for valid ids that are greater than 0, and will get the database. It will then attempt to delete the entire record where the stored id matches with the id in the database `Shazam-Artists` and will flash a message "Delete successful" to show the deletion was successful. If not, then a flash would appear showing that an error occurred deleting the record. We then unset that id and bring the user back to the admin list page after deleting.

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:

The delete logic here is a hard delete meaning it completely deletes the entity and its corresponding information from the table and would need to be either fetched again or custom created in order to be able to include that same data again in the table. The only roles that are able to delete the data from the database table is the admin role since it would be detrimnetal for the user to be deleting the data in the table that is shown to everyone. Future uses for regular users could include just removing the data from a watchlist for each respective user that wants to do that but will not be able to delete the entity from the entire database.

End of Task 1

Task

100%

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

COLLAPSE

Sub-Task

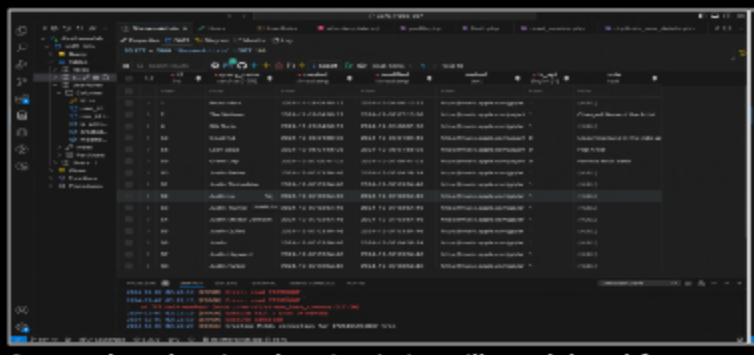
100%

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

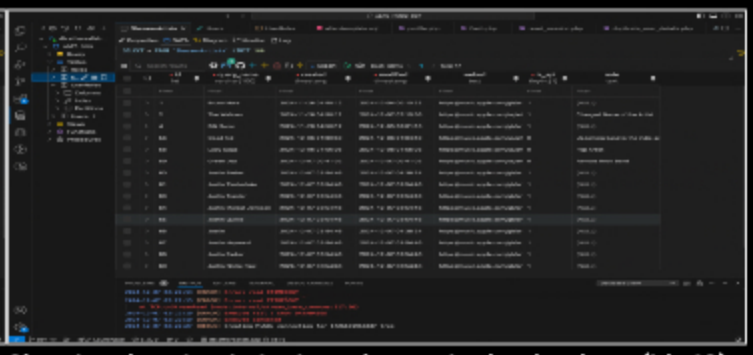
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Screenshot showing that Justin Lo will get deleted from database(id=62).



Showing that Justin Lo is no longer in the database(id=62).

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://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

https://it202-ob75-prod-62037cdcca6c.herokuapp.com/project/admin/list_artists.php

URL #2 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/49

URL #3 https://github.com/OmarBarrera1/ob75-IT202-007

https://github.com/OmarBarrera1/ob75-IT202-007/pull/54

End of Task 3

End of Group: Delete Handling

Task Status: 3/3

Group

100%

Group: Misc
Tasks: 4
Points: 1

COLLAPSE

Task

100%

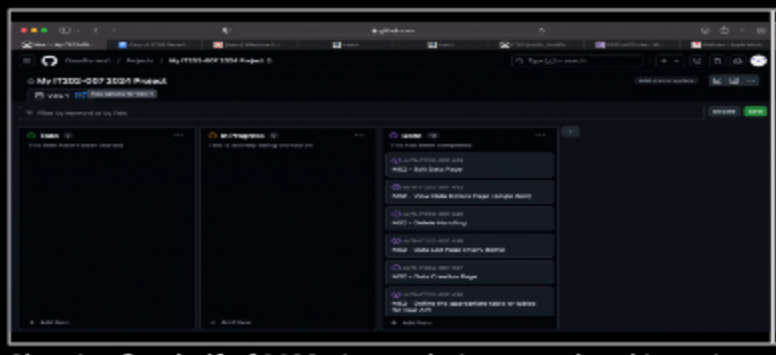
Group: Misc
Task #1: Screenshot of your project board from GitHub (tasks should be in the proper column)
Weight: ~25%
Points: ~0.25

COLLAPSE

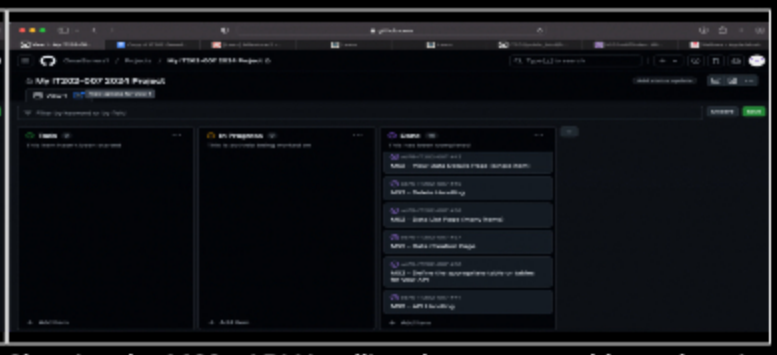
Task Screenshots

Gallery Style: 2 Columns

4 2 1



Showing first half of MS2 - issues being completed in project board in GitHub.



Showing the MS2 - API Handling that was not able to show in first screenshot from 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/OmarBarrera1/projects/1/views/1?layout=board

https://github.com/users/OmarBarrera1/projects/

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:

Some learnings that were from this assignment was learning how to properly map the API data from the API to the database which was one of the most important issues that were at hand when first beginning this assignment. The proper data from the API needed to have its parameter that it takes to match the parameter on the server side to be able to properly have each column in the database be correlated with the correct data from the API. An issue that arose was seeing how to make the differentiate the API fethced data with the custom data that the user would enter in and that was solved with the use of creating a new column in the table called is_api, in which defaulted to 1 when it was api data and went to 0 when it was custom data. Some other learnings came through the creation of the list page and creating all the filter functions to be able to filter the data from the database and search for specific data and limiting it through sorts.

End of Task 3

Task

100%

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

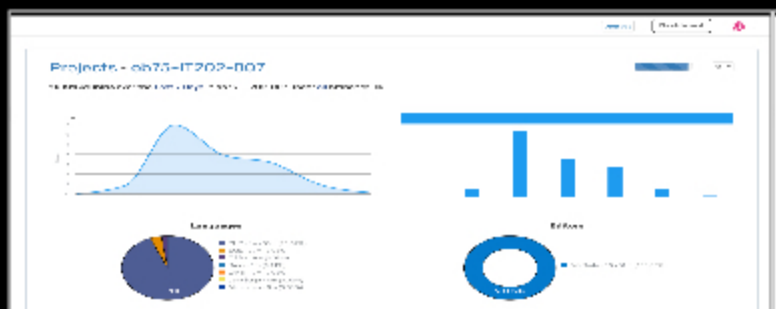
COLLAPSE

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



Showing Top half of WakaTime

Showing bottom half of Wakatime part1

Showing rest of bottom half of Wakatime part2

End of Task 4

End of Group: Misc
Task Status: 4/4

End of Assignment