Submission Worksheet

Submission Data

Course: IT202-450-M2025
Assignment: IT202 Milestone 2
Student: Mark Y. (may23)
Status: Submitted | **Worksheet Progress:** 100%
Potential Grade: 10.00/10.00 (100.00%)
Received Grade: 0.00/10.00 (0.00%)
Started: 7/21/2025 10:44:21 AM
Updated: 7/25/2025 12:43:29 PM
Grading Link: https://learn.ethereallab.app/assignment/v3/IT202-450-M2025/it202-milestone-2/grading/may23
View Link: https://learn.ethereallab.app/assignment/v3/IT202-450-M2025/it202-milestone-2/view/may23

Instructions

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

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

Section #1: (2 pts.) Data

Progress: 100%

≔ Task #1 (0.50 pts) - API Data Table

Progress: 100%

Part 1:

Progress: 100%

Details:

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

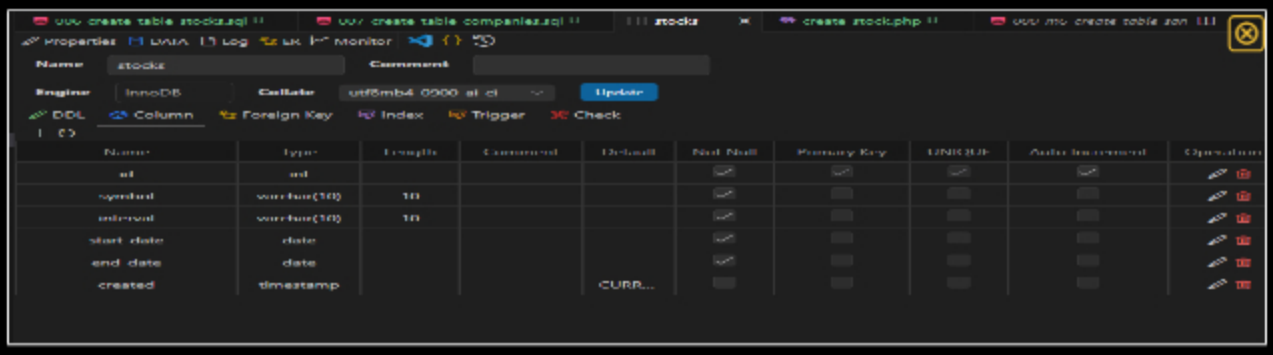
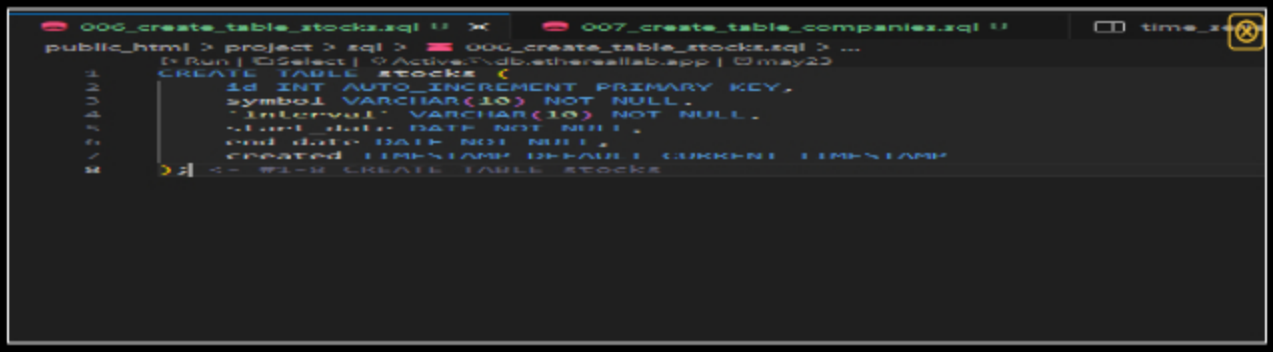


table structure



data table

Saved: 7/21/2025 10:51:45 AM

Part 2:

Progress: 100%

Details:

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

URL #1

https://github.com/Mark-5555/may23-IT202-450/Milestone2/public_html/project/sql/006_create_table_stocks.sql

URL

https://github.com/Mark-5555/ma

Saved: 7/21/2025 10:51:45 AM

Part 3:

Progress: 100%

Details:

- Briefly note what each field/property represents

Your Response:

- id: A unique auto-incrementing identifier for each stock record.
- symbol: The ticker symbol representing the stock (e.g., "AAPL" for Apple).
- interval: The time interval of the stock data (e.g., 1min, 1day), indicating how often the price is recorded.
- start_date: The beginning date of the time range for which data is being stored or requested.
- end_date: The end date of the data range.
- created: A timestamp automatically set when the row is inserted, tracking when the stock record was added to the database.

Saved: 7/21/2025 10:51:45 AM

Task #2 (0.50 pts.) - Data Transformation

Progress: 100%

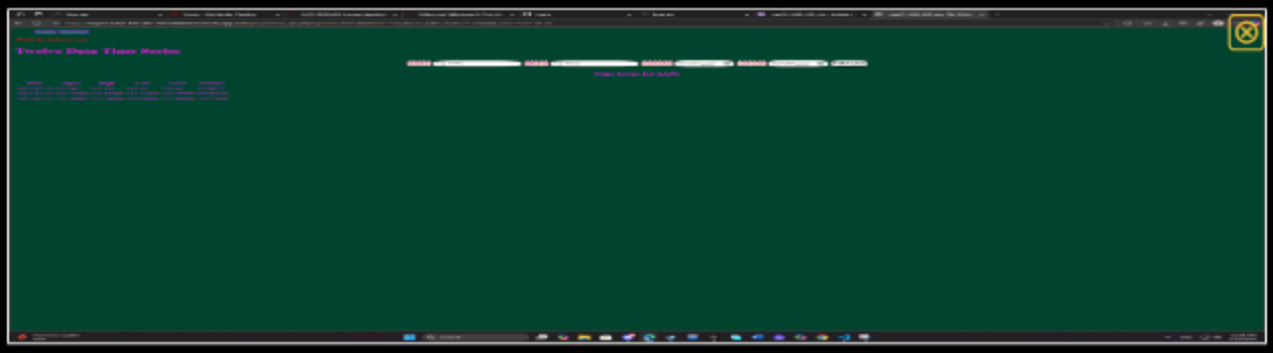
Part 1:

Progress: 100%

Details:

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

transformation code



heroku example

Saved: 7/25/2025 12:30:11 PM

Part 2:

Progress: 100%

Details:

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

URL #1
https://may23-it202-450-prod-70e1264ec6c5.herokuapp.com/project/test_api.php?symbol=AAPL&interval=1day&start_date=2025-07-21&end_date=2025-07-24

URL
https://may23-it202-450-prod-70e

URL #2
https://github.com/Mark-5555/may23-IT202-450

URL
https://github.com/Mark-5555/ma

Saved: 7/25/2025 12:30:11 PM

Part 3:

Progress: 100%

Details:

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

Your Response:

I will be using only a subset of the data returned from the API — specifically the fields relevant to my application: symbol, name, exchange, currency, country, and type.

I do not need to convert the data types, but I do extract only the needed properties from the full API response, which includes additional metadata.

The transformation logic involves decoding the JSON response and looping through the array of stock entries to build a simplified structure that retains just the selected fields, making the data easier to store in the database and use in the frontend.

Saved: 7/25/2025 12:30:11 PM

Task #3 (0.50 pts.) - API Duplicates

Progress: 100%

Details:

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

Your Response:

To handle duplicates, the application should check if the entity already exists in the database before inserting. If the API returns the same result for an entity that already exists and hasn't changed, the system should skip the insert to avoid redundancy. However, if the result has changed, the existing record should be updated instead of inserted. This can be done using an INSERT ... ON DUPLICATE KEY UPDATE SQL statement or by checking manually with a SELECT followed by an UPDATE. This ensures that the database remains accurate and avoids unnecessary duplication.

Saved: 7/23/2025 12:03:25 PM

Task #4 (0.50 pts.) - API Fetching

Progress: 100%

Details:

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

Your Response:

In my application, the data fetch is admin-triggered via a specific page (such as create_stock.php), where the admin can manually initiate the API request to fetch and store stock data. If the API returns the same result for an entity that already exists but has been modified, the application should detect the change by comparing the incoming data with the existing database record. If differences are found, the system will perform an update on the existing entry to ensure the stored data remains current and accurate, preventing stale information from being used or displayed.

Saved: 7/23/2025 12:04:27 PM

Section #2: (1.5 pts.) Data Creation

Progress: 100%

Task #1 (0.60 pts.) - Validations

Progress: 100%

Part 1:

Progress: 100%

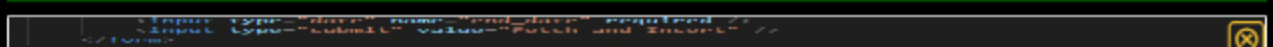
Details:

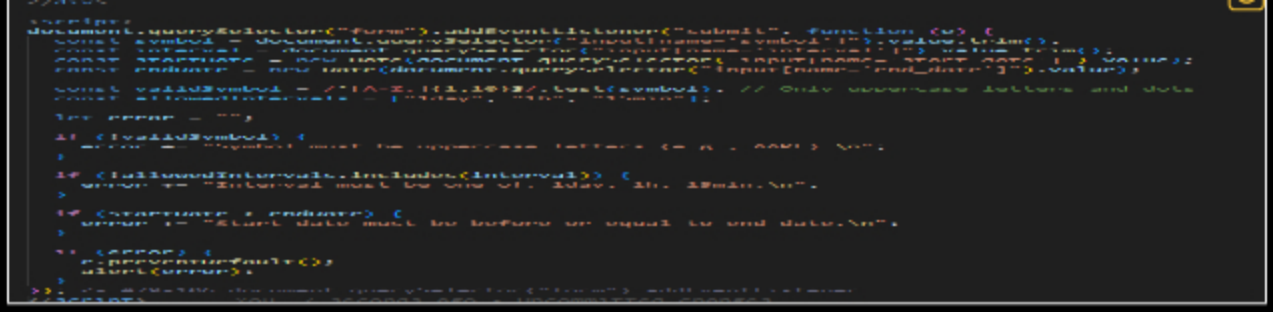
- Show the html validations (code)
- Show the javascript validations (code)
- Show the php validations (code)

```
div class="container-fluid">
    <h1>Insert Stock from API</h1>
    <form method="POST">
        <label>Symbol</label>
        <input name="symbol" required placeholder="e.g. AAPL" />
        <label>Interval</label>
        <input name="interval" required placeholder="e.g. 1day" />
        <label>Start Date</label>
        <input type="date" name="start_date" required />
        <label>End Date</label>
        <input type="date" name="end_date" required />
        <input type="submit" value="Fetch and Insert" />
    </form>
/div>
```

HTML validation code

PHP validation code





JS validation code

Saved: 7/23/2025 12:18:38 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

HTML validation is implemented using the required attribute on each field, ensuring that users must provide values for symbol, interval, start_date, and end_date before the form can be submitted.

JavaScript validation enhances this by checking that the symbol is in uppercase letters (e.g., "AAPL"), the interval is one of the allowed options (1day, 1h, or 15min), and that the start date is not after the end date; if any check fails, the form is prevented from submitting and an alert shows the errors.

On the PHP side, server-side validation uses isset() to confirm that all required $_POST fields are present before proceeding with database insertion and API calls, protecting the backend from incomplete or bypassed client inputs.

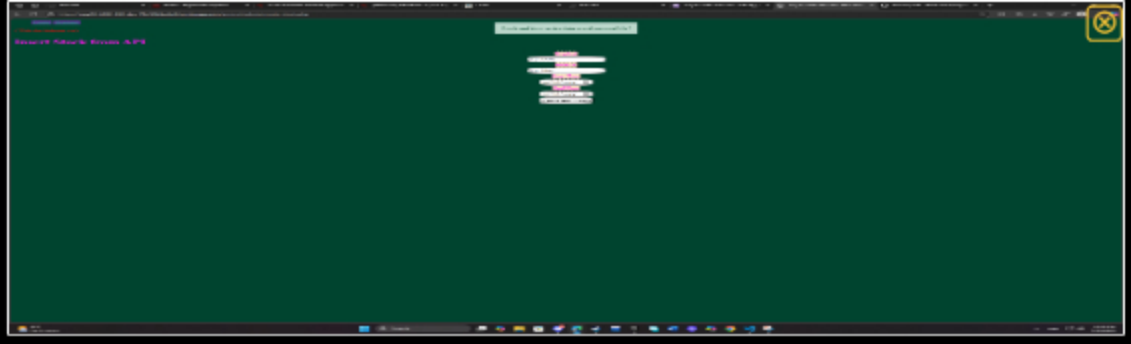
Saved: 7/23/2025 12:18:38 PM

Task #2 (0.60 pts.) - Examples

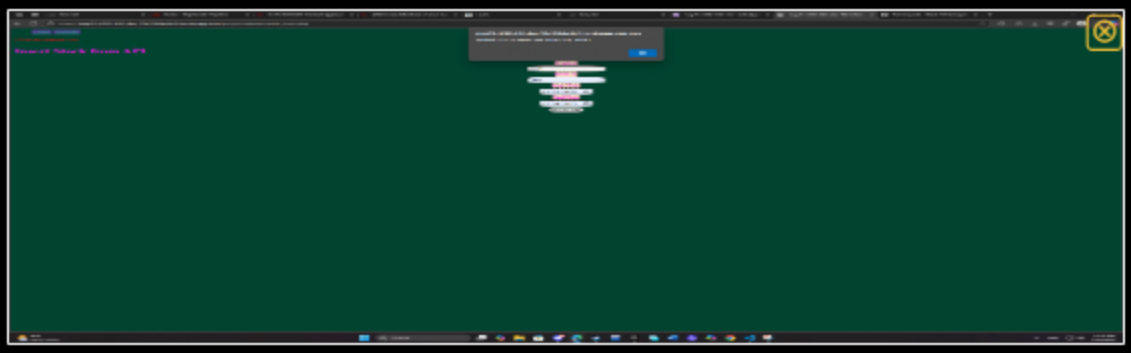
Progress: 100%

Details:

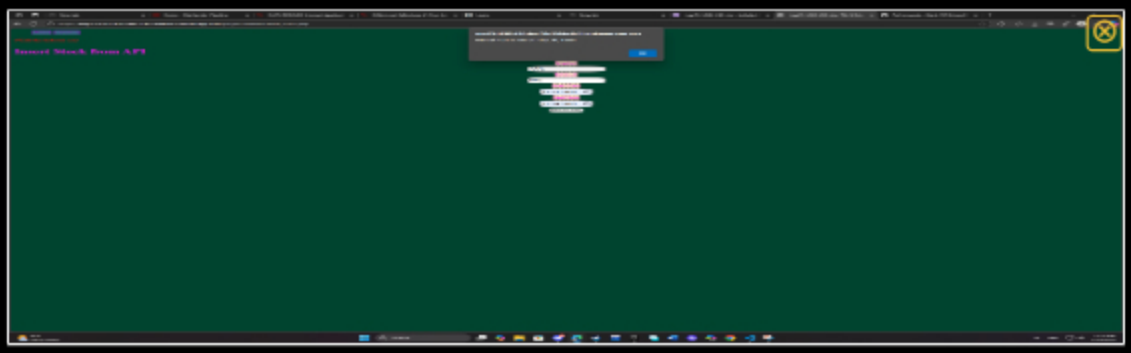
- Show a successful creation
- Demonstrate various validation errors
- Ensure heroku url is visible



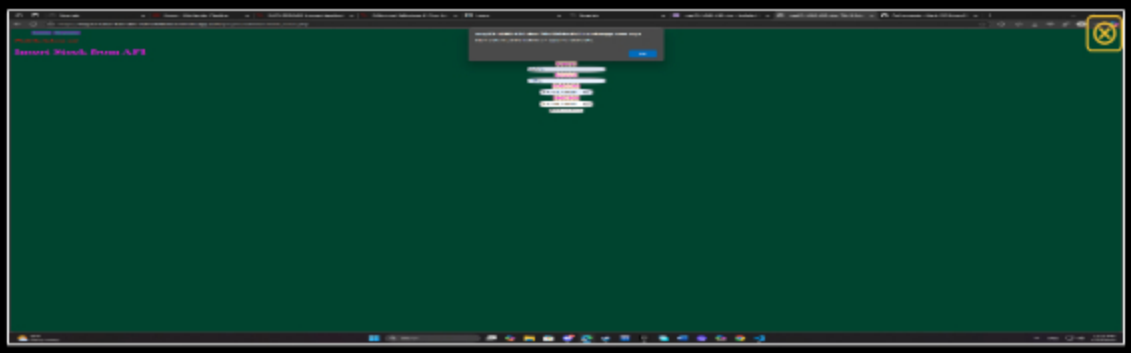
successful creation



lowercase name error



Invalid Interval



Start Date After End Date

Saved: 7/23/2025 12:26:01 PM

Task #3 (0.30 pts.) - Links

Progress: 100%

Details:

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

URL #1
https://may23-it202-450-prod-a1263d847137.herokuapp.com//project/admin/create_stock.php

URL
https://may23-it202-450-proc

URL #2
https://github.com/Mark-5555/may23-IT202-450/pull/35

URL
https://github.com/Mark-555

Saved: 7/25/2025 12:37:40 PM

Section #3: (1.5 pts.) Data List Page (Many Items)

Progress: 100%

Task #1 (0.60 pts.) - Code

Progress: 100%

Part 1:

Progress: 100%

Details:

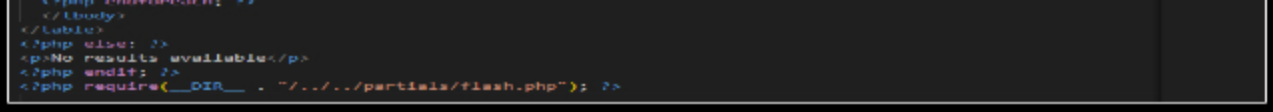
- Show the code that handles the filter/sort logic
- Show the code that generates the list/grid output
- Styling must be applied

```php
$limit = isset($_GET["limit"]) && is_numeric($_GET["limit"]) && $_GET["limit"] > 0 && $_GET["limit"] <= 10
$symbol = isset($_GET["symbol"]) ? $_GET["symbol"] : "";

$query = "SELECT * FROM stocks WHERE 1=1";
$params = [];
if ($symbol) {
    $query .= " AND symbol LIKE :symbol";
    $params[":symbol"] = "%$symbol%";
}
$query .= " ORDER BY created DESC LIMIT $limit";
$stmt = $db->prepare($query);
$stmt->execute($params);
```

filter code

```php
<?php if ($results): ?>
<table>
  <thead><tr><th>Symbol</th><th>Interval</th><th>Dates</th><th>Actions</th></tr></thead>
  <tbody>
  <?php foreach ($results as $row): ?>
    <tr>
      <td><?= htmlspecialchars($row["symbol"]) ?></td>
      <td><?= htmlspecialchars($row["interval"]) ?></td>
      <td><?= $row["start_date"] ?> - <?= $row["end_date"] ?></td>
<td>
  <a href="admin/view_stock.php?id=<?= $row["id"] ?>">View</a> |
  <a href="admin/edit_stock.php?id=<?= $row["id"] ?>">Edit</a> |
  <a href="admin/delete_stock.php?id=<?= $row["id"] ?>" onclick="return confirm('Delete this record?')">De
</td>
    </tr>
```



list code

Saved: 7/25/2025 1:36:20 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The filter/sort logic works by capturing the symbol and limit values from the URL query parameters, validating them, and using them to dynamically construct an SQL query that filters stock records by symbol and limits the number of results shown, defaulting to 10 if no valid limit is provided. The query always sorts records by the created date in descending order.

The list/grid output is generated using a simple HTML

where each row represents a stock entry with columns for the symbol, interval, date range, and action links (View, Edit, Delete), all populated using a PHP loop over the database results.

Styling choices are minimal and default to basic HTML elements, with no custom CSS applied; however, the table structure and form inputs provide a clear and organized layout suitable for listing records efficiently.

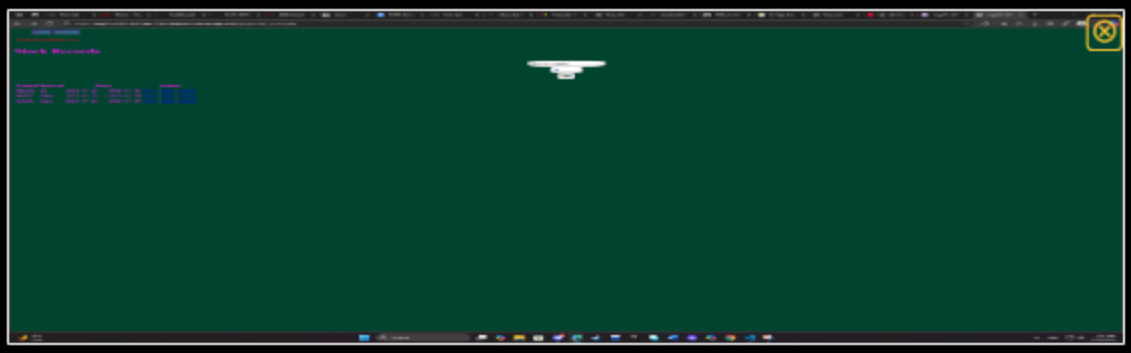
Saved: 7/25/2025 1:36:20 AM

Task #2 (0.60 pts.) - Examples

Progress: 100%

Details:

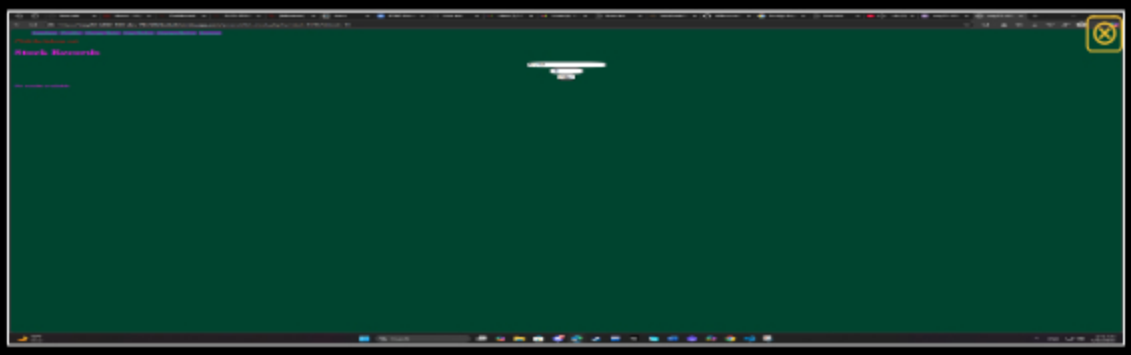
- Show a few variations of filter/sort applied (including no records)
- Each list item should be a summary
- Each list item should have the following links
  - A link to a single view of the specific entity (i.e., a details page)
  - A link to delete this entity (this may be an admin-only functionality, but it should be present for the respective role)
  - A link to edit this entity (this may be an admin-only functionality, but it should be present for the respective role)
- Ensure heroku url is visible



default (10)

filter by symbol

filter to 2



not found

Saved: 7/25/2025 1:46:46 AM

Task #3 (0.30 pts.) - Links

Progress: 100%

Details:

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

URL #1
https://github.com/Mark-5555/may23-IT202-450/pull/42

URL
https://github.com/Mark-555

URL #2
https://may23-it202-450-prod-a1263d847137.herokuapp.com//project/list_stock.php

URL
https://may23-it202-450-proc

Saved: 7/25/2025 12:37:45 PM

Section #4: (1.5 pts.) View Details Page (Single Item)

Progress: 100%

Task #1 (0.60 pts.) - Code

Progress: 100%

Part 1:

Progress: 100%

Details:

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

```php
$id = $_GET["id"] ?? null;
if (!$id || !is_numeric($id)) {
    flash("Invalid ID");
    redirect("list_stock.php");
}

$db = getDB();
$stmt = $db->prepare("SELECT * FROM stocks WHERE id = :id");
$stmt->execute([":id" => $id]);
$stock = $stmt->fetch(PDO::FETCH_ASSOC);
if (!$stock) {
    flash("Stock not found");
    redirect("list_stock.php");
}
?>
```

loading the record code

```php
$id = $_GET["id"] ?? null;
if (!$id || !is_numeric($id)) {
    flash("Invalid ID");
    redirect("list_stock.php");
}
```

```php
$db = getDB();
$stmt = $db->prepare("SELECT * FROM stocks WHERE id = :id");
$stmt->execute([":id" => $id]);
$stock = $stmt->fetch(PDO::FETCH_ASSOC);
if (!$stock) {
    flash("Stock not found");
    redirect("list_stock.php");
}
?>
```

invalid or missing id code

```php
<h1>Stock Details</h1>
<ul>
    <li><strong>Symbol:</strong> <?= htmlspecialchars($stock["symbol"]) ?></li>
    <li><strong>Interval:</strong> <?= htmlspecialchars($stock["interval"]) ?></li>
    <li><strong>Date Range:</strong> <?= $stock["start_date"] ?> to <?= $stock["end_date"] ?></li>
</ul>
<a href="../edit_stock.php?id=<?= $stock["id"] ?>">Edit</a> |
<a href="../delete_stock.php?id=<?= $stock["id"] ?>" onclick="return confirm('Are you sure?')">Delete</a>

<?php require(__DIR__ . "/../../partials/flash.php"); ?>
```

generate entity code

Saved: 7/25/2025 11:11:59 AM

Part 2:

Progress: 100%

Details:

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

Your Response:

The loading logic works by first retrieving the id parameter from the URL using $_GET[id], then validating that it's present and numeric. If valid, a prepared SQL statement is executed to securely fetch the stock record from the database using that id. If the ID is missing, invalid, or the record doesn't exist, a flash message is shown and the user is redirected to the listing page.

The entity output is generated using an unordered list that displays the symbol, interval, and date range of the stock with clear labels, and includes "Edit" and "Delete" action links beneath.

The styling is simple and functional, using basic HTML elements like ul, li, and strong for readability, with no additional CSS or layout frameworks applied.

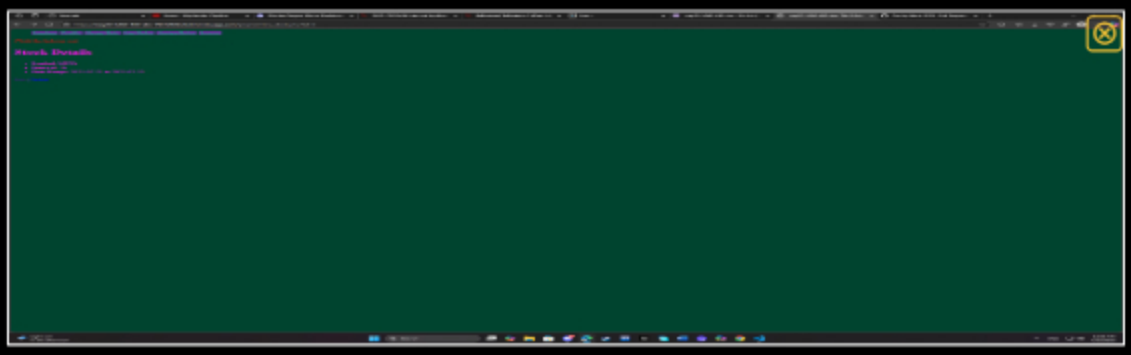
Saved: 7/25/2025 11:11:59 AM

Task #2 (0.60 pts.) - Examples

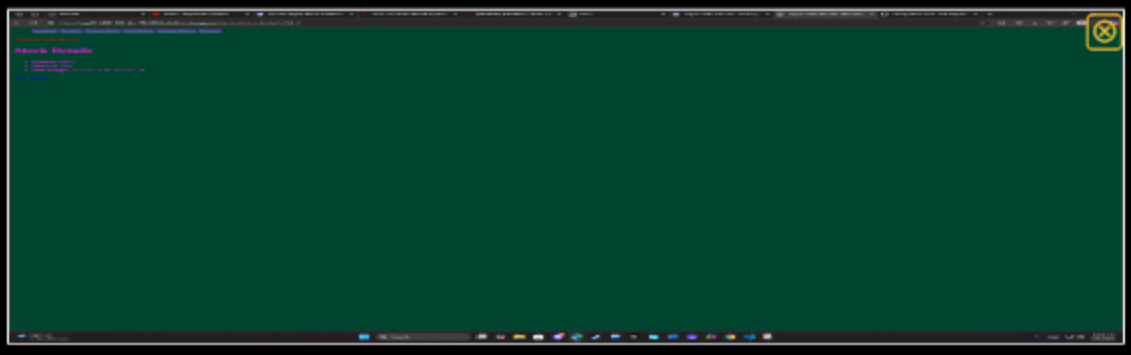
Progress: 100%

Details:

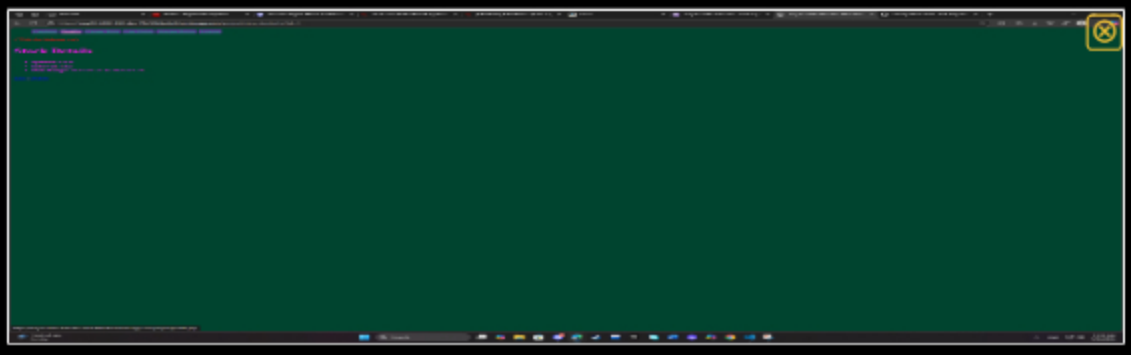
- Show a few examples of different entities
- Each entity item should have more details than the summary view (if possible)
- Each entity should have the following links
  - A link to edit the single entity (this may be an admin-only thing, but it should be present for the respective role)
  - A link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



example 1



example 2



example 3

Saved: 7/25/2025 11:33:56 AM

Task #3 (0.30 pts.) - Links

Progress: 100%

Details:

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

URL #1
https://may23-it202-450-prod-a1263d847137.herokuapp.com//project/view_stock.php?id=5

URL
https://may23-it202-450-prod

URL #2
https://github.com/Mark-5555/may23-IT202-450/pull/42

URL
https://github.com/Mark-555

Saved: 7/25/2025 12:37:50 PM

Section #5: (1.5 pts.) Edit Page

Progress: 100%

Task #1 (0.60 pts.) - Code

Progress: 100%

Part 1:

Progress: 100%

Details:

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

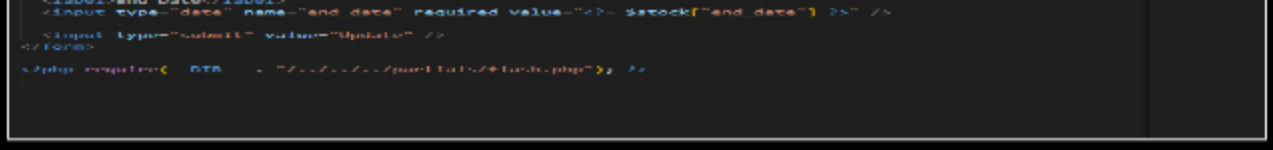
```php
$id = $_GET["id"] ?? null;
if (!$id || !is_numeric($id)) {
    flash("Invalid ID");
    redirect("list_stocks.php");
}

$db = getDB();
$stmt = $db->prepare("SELECT * FROM stocks WHERE id = :id");
$stmt->execute([":id" => $id]);
$stock = $stmt->fetch(PDO::FETCH_ASSOC);
if (!$stock) {
    flash("Stock not found");
    redirect("../list_stock.php");
}
```

You, 2 days ago • adding Milestone2 files ...

loading of a record code

```
<h1>Edit Stock</h1>
<form method="POST">
    <label>Symbol</label>
    <input name="symbol" required pattern="[A-Z.]{1,10}" title="1-10 uppercase letters or dots" value="<?= h
    <label>Interval</label>
    <input name="interval" required list="interval-options" value="<?= htmlspecialchars($stock['interval']) ?>" />
    <datalist id="interval-options">
        <option value="1day" />
        <option value="1h" />
        <option value="1min" />
    </datalist>
    <label>Start Date</label>
    <input type="date" name="start_date" required value="<?= $stock['start_date'] ?>" />
```



HTML form code

php code

JS code

Saved: 7/25/2025 12:21:03 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The loading logic in edit_stock.php begins by retrieving the id from the URL using $_GET; it validates whether it's numeric and exists. If invalid or not found in the database, the user is redirected with an appropriate flash message. Once validated, a SELECT query fetches the record for prefilling the form.

The form is generated using appropriate HTML input types (text for symbol and interval, date for start and end dates) and is pre-filled using values from the database. Submission is handled via

POST, with JavaScript validating empty fields and date order, and PHP performing final server-side checks before updating the database.

The page uses basic form styling with clear labels and fields stacked vertically for readability. Only the fields symbol, interval, start_date, and end_date are editable — id, created, and modified are omitted intentionally to preserve data

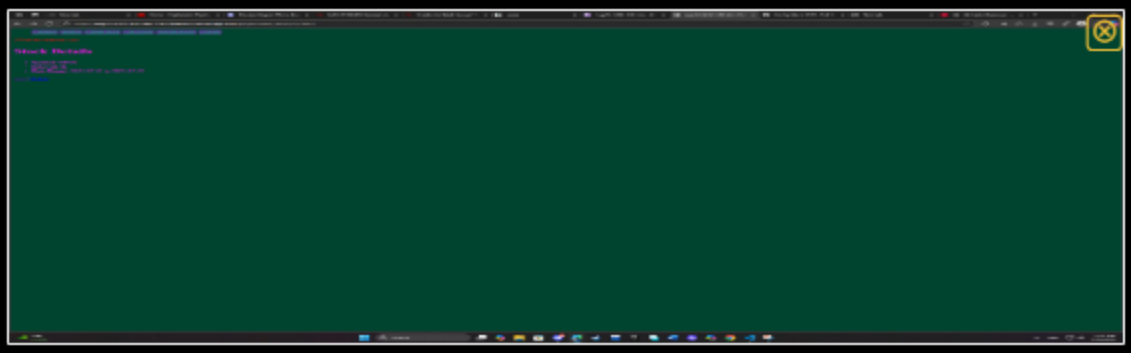
Saved: 7/25/2025 12:21:03 PM

Task #2 (0.60 pts.) - Examples

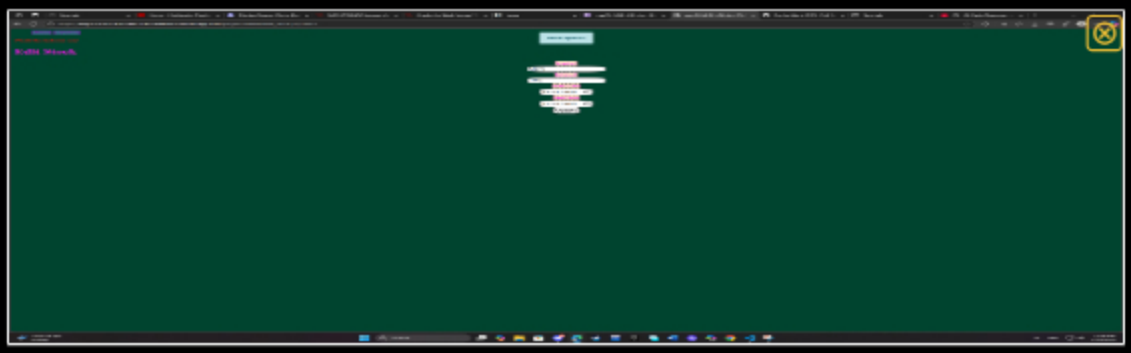
Progress: 100%

Details:

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

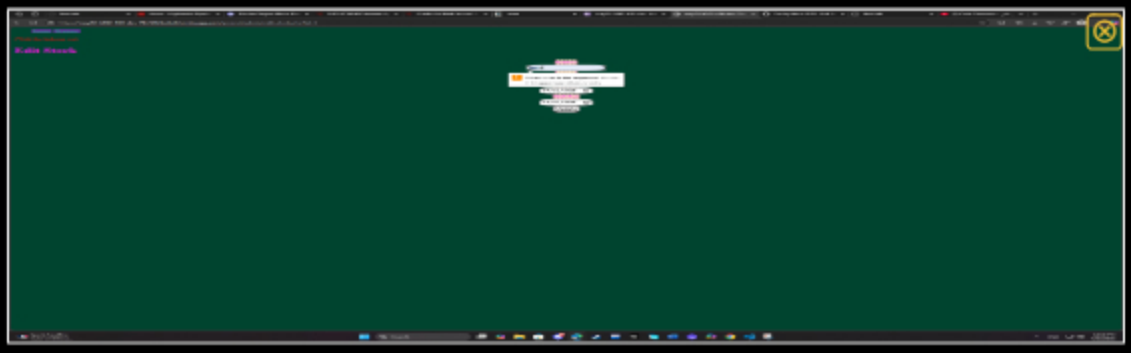


before

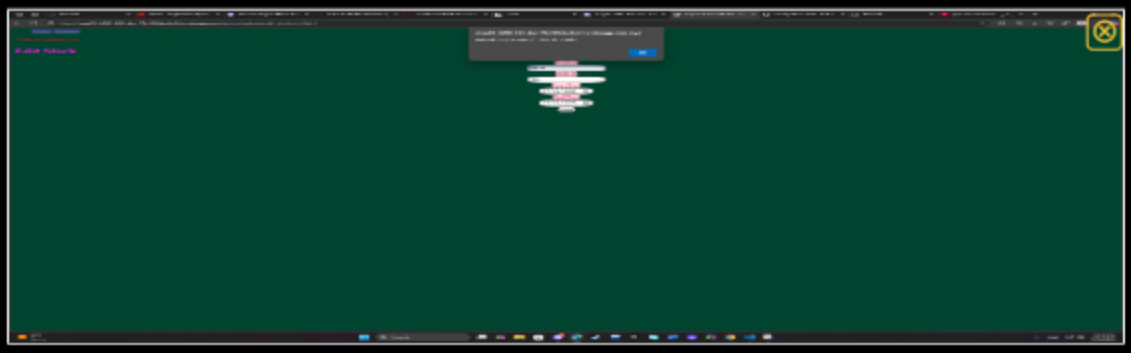


successful update message

after



invalid symbol



invalid interval

Saved: 7/25/2025 12:19:50 PM

Task #3 (0.30 pts.) - Links

Progress: 100%

Details:

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

URL #1
https://may23-it202-450-prod-a1263d847137.herokuapp.com//project/admin/edit_stock.php?id=5

URL
https://may23-it202-450-proc

URL #2
https://github.com/Mark-5555/may23-IT202-450/pull/50

URL
https://github.com/Mark-555

Saved: 7/25/2025 12:37:55 PM

Section #6: (1.5 pts.) Delete Logic

Progress: 100%

Task #1 (0.60 pts.) - Code

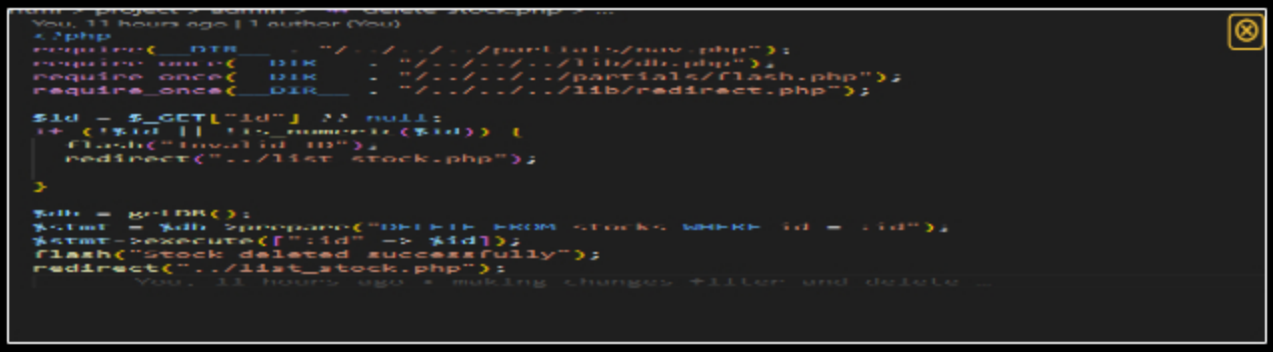
Progress: 100%

Part 1:

Progress: 100%

Details:

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



delete code& redirect logic

Saved: 7/25/2025 12:25:21 PM

Part 2:

Progress: 100%

Details:

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

Your Response:

The deletion logic uses a hard delete by executing a DELETE FROM stocks WHERE id = :id query after validating the id from the GET request. This permanently removes the stock record from the database (not a soft delete).

There are no specific permissions or role checks directly in the script, so unless protected elsewhere, any user could access it, which makes it important to restrict deletion to admins.

After deletion, a success message is flashed and the user is redirected to list_stock.php; however, the redirect does not retain any previous filter or sort parameters, so the original view state is lost after deletion.

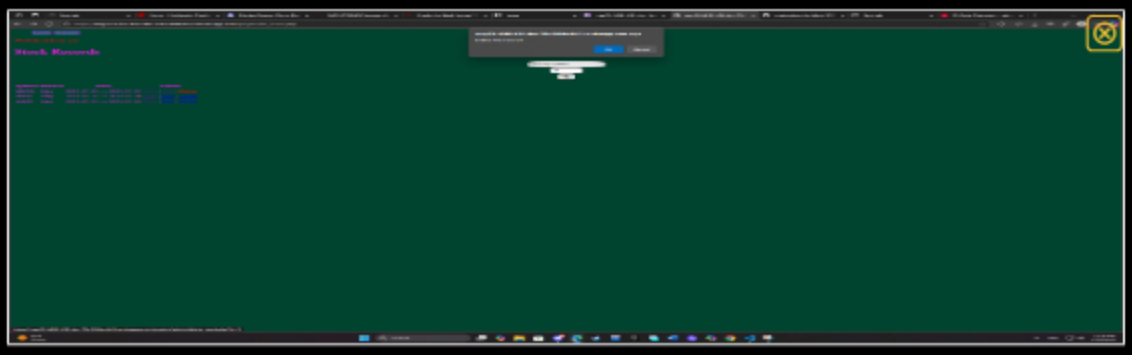
Saved: 7/25/2025 12:25:21 PM

Task #2 (0.60 pts.) - Examples

Progress: 100%

Details:

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



delete confim message

delete message& updated list

Saved: 7/25/2025 12:28:08 PM

Task #3 (0.30 pts.) - Links

Progress: 100%

Details:

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

URL #1
https://may23-it202-450-prod-a1263d847137.herokuapp.com//project/admin/delete_stock.php?id=5

URL
https://may23-it202-450-proc

URL #2
https://github.com/Mark-5555/may23-IT202-450/pull/42

URL
https://github.com/Mark-555

Saved: 7/25/2025 12:37:59 PM

Section #7: (0.5 pts.) Misc

Progress: 100%

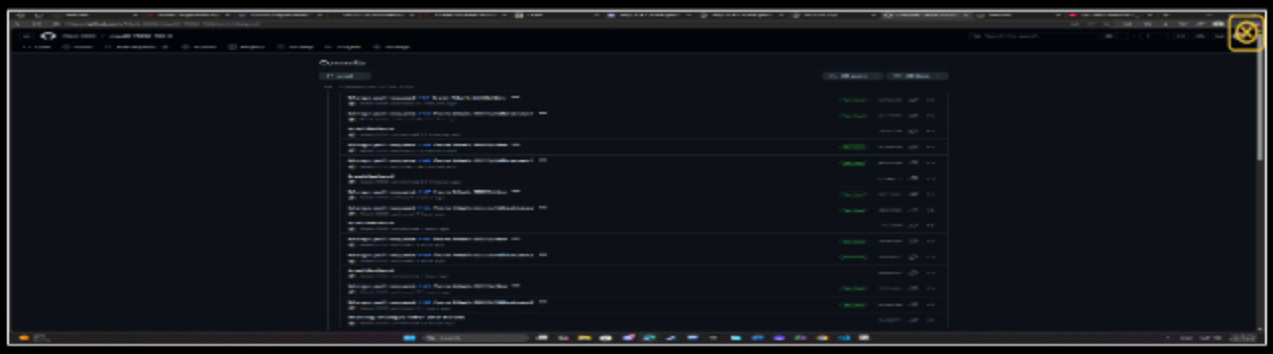
Task #1 (0.17 pts.) - Github Details

Progress: 100%

Part 1:

Progress: 100%

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



commits history

Saved: 7/25/2025 12:39:49 PM

Part 2:

Progress: 100%

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

URL #1
https://github.com/Mark-5555/may23-IT202-450

URL
https://github.com/Mark-5555/ma

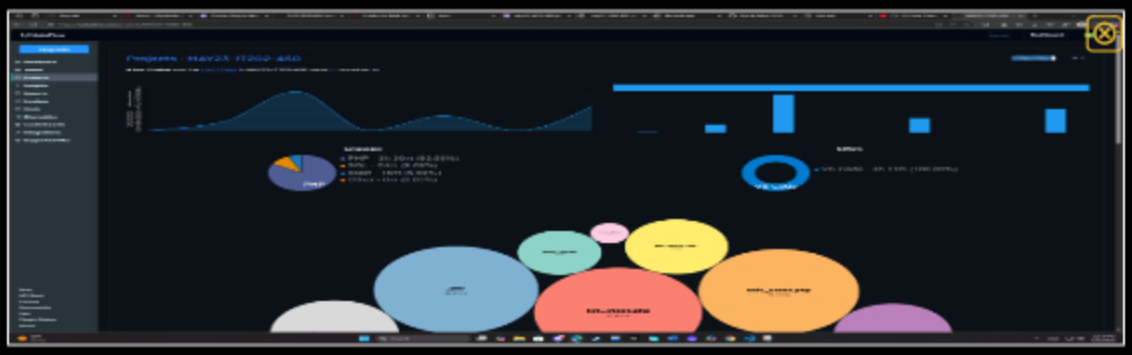
Saved: 7/25/2025 12:39:49 PM

Task #2 (0.17 pts.) - WakaTime - Activity

Progress: 100%

Details:

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



wakatime top

wakatime bottom

Saved: 7/25/2025 12:41:16 PM

Task #3 (0.17 pts.) - Reflection

Progress: 100%

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

Progress: 100%

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

Your Response:

From working on Milestone 2, I learned how to build a full CRUD system in PHP integrated with a real-world API. I practiced HTML, JavaScript, and PHP validations, how to manage GET/POST requests securely, how to handle and display user feedback using flash messages, and how to dynamically load and update data from a MySQL database. I also saw how role-based access, foreign key constraints, and redirect logic play a critical role in multi-page apps.

Saved: 7/25/2025 12:42:41 PM

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

Progress: 100%

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

Your Response:

The easiest part was creating the HTML forms and handling basic form submissions. Once the structure was in place, setting up required fields and passing values through POST was straightforward. Displaying data in the list view with a simple foreach loop was also easy.

Saved: 7/25/2025 12:43:20 PM

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

Progress: 100%

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

Your Response:

The hardest part was debugging routing and path errors, especially when mixing different folder depths like admin/ vs project/. Ensuring the redirect paths, flash messages, and required includes worked properly across different directories took extra time and attention. Handling database constraints (like foreign keys preventing deletions) was also a bit tricky without clear error messaging.

Saved: 7/25/2025 12:43:29 PM