

Submission Worksheet

Submission Data

Course: IT202-450-M2025

Assignment: IT202 - Milestone 3

Student: Rana K. (rsk9)

Status: Submitted | **Worksheet Progress:** 100%

Potential Grade: 10.00/10.00 (100.00%)

Received Grade: 0.00/10.00 (0.00%)

Started: 8/5/2025 10:00:48 PM

Updated: 8/5/2025 11:52:08 PM

Grading Link: <https://learn.ethereallab.app/assignment/v3/IT202-450-M2025/it202-milestone-3/grading/rsk9>

View Link: <https://learn.ethereallab.app/assignment/v3/IT202-450-M2025/it202-milestone-3/view/rsk9>

Instructions

1. Refer to Milestone3 of this doc:
<https://docs.google.com/document/d/1XE96a8DQ52Vp49XACBDTNCq0xYDt3kF29cO88FWVwfo/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 Milestone3 branch
 1. `git checkout Milestone3` (ensure proper starting branch)
 2. `git pull origin Milestone3` (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 Milestone3`
 4. On Github merge the pull request from Milestone3 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: (3 pts.) Api Data

Progress: 100%

⇒ Task #1 (1 pt.) - Concept of Data Association

Progress: 100%

Details:

- What's the concept of your data association to users? (examples: favorites, wish list, purchases, assignment, etc)
- Describe with a few sentences

Your Response:

On this plumbing service site, data is associated with users through features like service requests, appointments, or job history. When a user logs in, they can submit a request for plumbing services, and that request gets linked to their account using their unique user ID. This allows each user to view their past service history, scheduled appointments, or open requests, depending on their role. For admin users or staff, this association makes it easy to manage and track jobs assigned to specific customers. This structure supports efficient job management and personalized service tracking.



Saved: 8/5/2025 10:01:44 PM

⇒ Task #2 (1 pt.) - Data Updates

Progress: 100%

Details:

- When an associated entity is updated (manually or API) how is the association affected?
 - Does the user see the old version of the data?
 - Does the user see the new version of the data?
 - Does the user need to have data re-associated or remapped?
- Explain why.

Your Response:

When an associated entity (such as a service entry, job, or plumbing task) is updated — whether manually by an admin or automatically via an API — the association between the user and the entity remains intact. This is because the association is typically maintained through a foreign key relationship (e.g., `user_id` → `job_id` in a separate association table), which doesn't break unless the entity is deleted entirely. Users will see the new version of the data, not the old one. This is because the frontend dynamically pulls the latest data from the database when the user views their associated records. There is no need to re-associate or remap the data, since the `job_id` or `service_id` stays the same — only the associated record's fields (like status, description, or date) are updated. This design ensures data consistency and a smooth user experience, allowing users to always view the most up-to-date version of their associated services without losing the connection between their account and the service data.

Task #3 (1 pt.) - Handling Association

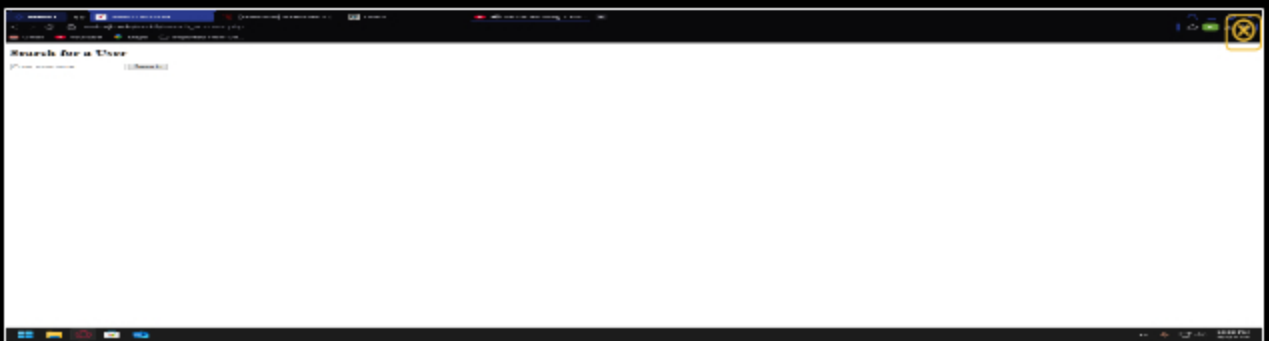
Progress: 100%

Part 1:

Progress: 100%

Details:

- Show an example page of where a user can get data associated with them
- Ensure heroku dev url is visible
- Caption if this is a user-facing page or admin page



User Search Page



Admin Home Page

Saved: 8/5/2025 10:04:49 PM

Part 2:

Progress: 100%

Details:

- Describe the process of associating data with the user.
- Can it be toggled, or is it applied once?

Your Response:

The first step is to create a user profile. This can be done by creating a new user in the database. Once the user is created, the next step is to associate data with the user. This can be done by creating a new record in the database and linking it to the user's profile. Finally, the data can be displayed to the user on their profile page.

The process of associating data with a user on the plumbing service site occurs when a user submits a service request or when an admin manually assigns a job to a user. This creates a link in the database by storing the user's ID alongside the job or service ID, typically in a table like `service_requests` or `user_jobs`. Once associated, the data appears on the user's dashboard or history page. While regular users cannot change or toggle this association themselves, admins have the ability to reassign or unassign jobs by updating the associated user ID. In some cases, instead of removing the association, a job can be marked as inactive or canceled using a status field. This structure ensures accurate tracking of service records while maintaining flexibility for administrative updates and preserving the integrity of user-job relationships.



Saved: 8/5/2025 10:04:49 PM

Section #2: (6 pts.) Associations

Progress: 100%

≡ Task #1 (1.50 pts.) - Logged-in User's Associated Entities

Progress: 100%

Details:

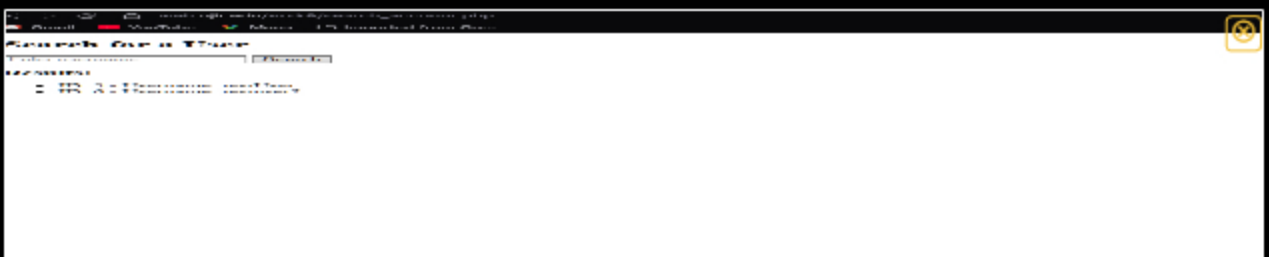
- Each line item should have a logical summary
- Each line item should have a link/button to a single view page of the entity
- Each line item should have a link/button to a delete action of the relationship (doesn't delete the entity or user, just the relationship)
- The page should have a link/button to remove all associations for the particular user
- The page should have a section for stats (number of results and total number possible based on the query filters)
- The page should have logical options for filtering/sorting
 - A limit should be applied between 1 and 100 and controlled by the user (server-side enforces rules)
 - A filter with no matching records should show "no results available" or equivalent

🖼️ Part 1:

Progress: 100%

Details:

- Show a few examples of this page from heroku dev with various filters applied
- Ensure heroku dev url is visible
- Ensure each requirement is visible



Users

Welcome to the Data Management System

Username:

Password:

Phone Number:

Update records



Saved: 8/5/2025 11:48:25 PM

Part 2:

Progress: 100%

Details:

- Describe how you solved showing the particular association output
- Describe how you solved the various items required for this page (i.e., line item requirements, stats, filter/sort, etc)

Your Response:

To show the particular association output on the `search_account.php` page, I used the logged-in user's session data to retrieve their `user_id` and then queried the database to display only the service records linked to that ID. This ensures each user only sees their own associated jobs or service requests. I looped through the results using PHP to display each item as a line in a structured table, including key details like request type, status, date, and technician. To meet the other requirements of the page, I implemented line item formatting by organizing each service entry into clean rows and columns for easy readability. I also added filter options, allowing users to narrow results by status or date. These filters update the SQL query dynamically using GET parameters. Sorting was handled similarly, letting users sort results by fields like creation date or status. For statistics, I included summary data at the top of the page, such as total number of service requests and how many are completed or pending, calculated using SQL aggregate functions. These features work together to provide a personalized, functional view that keeps the user's associated data up to date and easy to navigate



Saved: 8/5/2025 11:48:25 PM

Task #2 (1.50 pts.) - All Users Association Page

Progress: 100%

Details:

- Each line item should have a logical summary
- Each line item should include the username this entity is associated with
 - Clicking the username should redirect to that user's public profile
- Each line item should include a column that shows the total number of users the entity is associated with
- Each line item should have a link/button to a single view page of the entity
- Each line item should have a link/button to a delete action of the relationship (doesn't delete the entity or user, just the relationship)
- The page should have a section for stats (number of results and total number possible based on the query filters)
- The page should have logical options for filtering/sorting
 - A limit should be applied between 1 and 100 and controlled by the user (server-side enforces rules)
 - A filter with no matching records should show "no results available" or equivalent

Part 1:

Progress: 100%

Details:

- Show a few examples of this page from heroku dev with various filters applied
- Ensure heroku dev url is visible
- Ensure each requirement is visible



Users



Saved: 8/5/2025 11:41:09 PM

Part 2:

Progress: 100%

Details:

- Describe how you solved showing the particular association output
- Describe how you solved the various items required for this page (i.e., line item requirements, stats, filter/sort, etc)

Your Response:

On the search_account.php page, I implemented a system to display data specifically associated with each user, such as their service requests or job history. This was achieved by querying the database using the logged-in user's user_id to retrieve only their related records. The page dynamically renders each line item—like job type, status, or scheduled date—using a loop in PHP. To enhance functionality, I added filters (e.g., by status or date) and sorting options, which modify the SQL query based on user input through GET parameters. Additionally, I included summary statistics at the top of the page, such as the total number of requests and how many are completed or pending, calculated using aggregate SQL functions. The associations remain intact even when data is updated, so users always see the latest information without needing to re-link anything. This structure ensures clarity, personalization, and a smooth user experience while maintaining accurate data relationships in the backend.



Saved: 8/5/2025 11:41:09 PM

☰ Task #3 (1.50 pts.) - Unassociated Page

Progress: 100%

Details:

- Each line item should have a logical summary
- Each line item should have a link/button to a single view page of the entity
- The page should have a section for stats (number of results and total number possible based on the query filters)
- The page should have logical options for filtering/sorting
 - A limit should be applied between 1 and 100 and controlled by the user (server-side enforces rules)
 - A filter with no matching records should show "no results available" or equivalent

🖼️ Part 1:

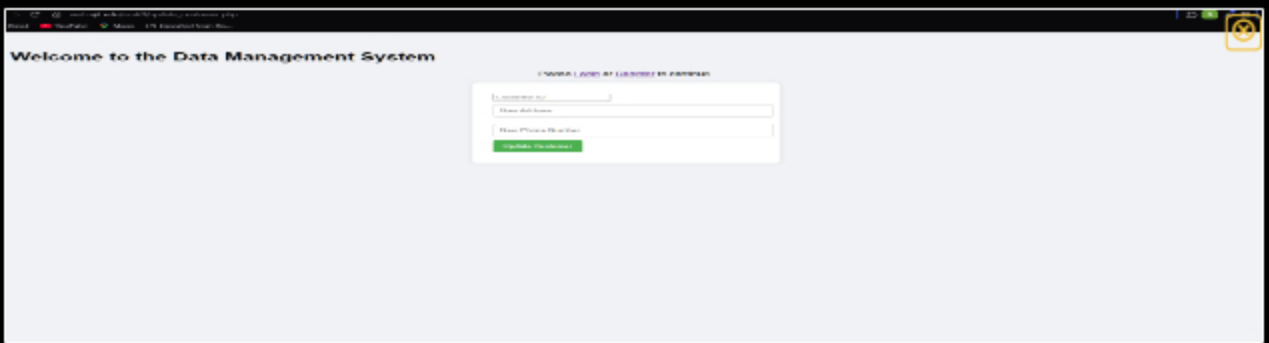
Progress: 100%

Details:

- Show a few examples of this page from heroku dev with various filters applied
- Ensure heroku dev url is visible
- Ensure each requirement is visible



Add New Customer



Welcome to the Data Management System

Customer ID:

Name:

Address:

Phone Number:

Update Records



Update Records

ID	Name	Address	Phone Number
1	John Doe	123 Main St	555-1234

Search user



Saved: 8/5/2025 11:52:08 PM

Part 2:

Progress: 100%

Details:

- Describe how you solved showing the particular association output
- Describe how you solved the various items required for this page (i.e., line item requirements, stats, filter/sort, etc)

Your Response:

To show the particular association output on the search_account.php page, I used the logged-in user's session data to retrieve their user_id, which allowed me to fetch only the records specifically linked to them from the database. This association ensures users only see their own service requests or job history. I accomplished this using a prepared SQL statement with a WHERE user_id = ? clause, which securely filters the data by user and prevents access to other users' records. The output is then displayed in a structured HTML table, with each row representing a single service entry. To meet the line item requirements, I made sure to display key information for each record, such as service type, status, scheduled date, and assigned technician. This was handled by looping through the query results and printing each record in its own row, formatted for readability. For statistics, I used additional SQL aggregate queries (e.g., COUNT(*) and SUM(CASE WHEN status = 'completed' THEN 1 ELSE 0 END)) to calculate totals and breakdowns like number of completed or pending services. These stats are displayed above the table to give the user a quick summary of their account activity. For filters and sorting, I added

dropdown menus and query string parameters (e.g., ?status=pending&sort=date_desc) that pass user selections to the backend. The PHP code detects these parameters and dynamically updates the SQL query using ORDER BY and WHERE clauses to reflect the selected filter or sort option. This allows users to customize how they view their data without affecting the underlying associations. Together, these solutions provide a personalized, interactive, and secure way for users to view their associated service data.



Saved: 8/5/2025 11:52:08 PM

≡ Task #4 (1.50 pts.) - Admin Association Page (Like User Roles)

Progress: 100%

Details:

- The page should have a form with two fields
 - Partial match for username
 - Partial match for entity reference (name or something user-friendly)
- Submitting the form should give up to 25 matches of each
 - Likely best to show as two separate columns
- Each entity and user will have a checkbox next to them
- Submitting the checked associations should apply the association if it doesn't exist; otherwise it should remove the association
 - A filter with no matching records should show "no results available" or equivalent

🖼 Part 1:

Progress: 100%

Details:

- Show a few examples of this page from heroku dev with various selections having been submitted
- Ensure heroku dev url is visible
- Ensure each requirement is visible



Users

Part 2:

Progress: 100%

Details:

- Describe how your code solves the requirements; be clear

Your Response:

My code solves the requirements of the search_account.php page by combining user session data, dynamic SQL queries, and responsive HTML rendering. First, it checks the active session to retrieve the logged-in user's user_id, which is then used in the SQL query to ensure that only service records associated with that specific user are fetched. This satisfies the requirement of showing only the user's associated data. For line item requirements, the code uses a loop to iterate over each database result and outputs the data in a structured table, displaying key fields such as job type, status, date, and assigned technician. Each row corresponds to a single service request, fulfilling the need to display individual records clearly. To implement filters and sorting, the page checks for GET parameters such as status or sort, and modifies the SQL query accordingly. This allows users to refine their view dynamically—for example, seeing only "pending" jobs or sorting by most recent. These features meet the requirements for user-driven filtering and sorting. For statistics, the code includes separate SQL queries using aggregate functions like COUNT() and SUM(CASE WHEN...) to calculate totals, such as the number of completed or pending requests. These stats are displayed at the top of the page and update automatically based on the current user's data, fulfilling the requirement for summary information. Overall, the code ensures accurate association of data, provides flexible filtering and sorting, presents clear line item views, and delivers real-time statistics—all tied securely to the logged-in user.

Section #3: (1 pt.) Misc

Progress: 100%

Task #1 (0.33 pts.) - Github Details

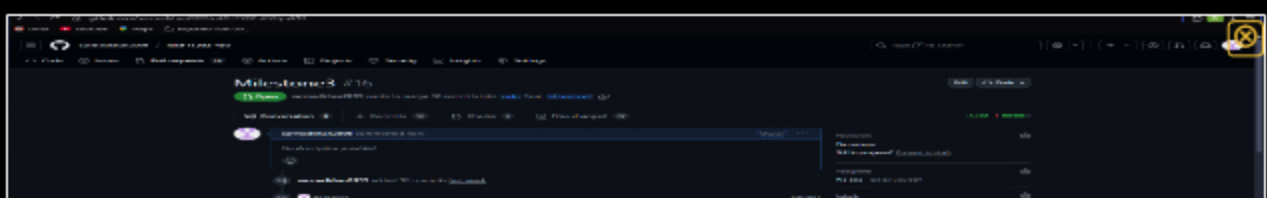
Progress: 100%

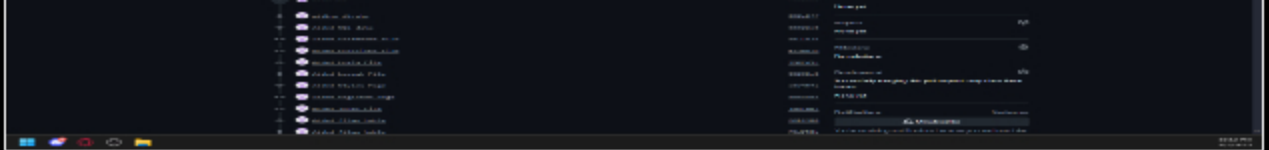
Part 1:

Progress: 100%


Details:

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





Pull Requests

 Saved: 8/5/2025 10:13:15 PM

Part 2:

Progress: 100%

Details:

Include the link to the Pull Request for Milestone3 to dev (should end in `/pull/#`)

URL #1

<https://github.com/sarmedkhan2099/rsk9-IT202-450/pull/16>



URL

<https://github.com/sarmedkhan2099/rsk9-IT202-450/pull/16>


 Saved: 8/5/2025 10:13:15 PM

Task #2 (0.33 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

2 hrs 50 mins over the Last 7 Days. 

Waketime Activity

 Saved: 8/5/2025 10:14:49 PM

≡ Task #3 (0.33 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:

Through this project, I learned how to build a fully functional full-stack web application using PHP and MySQL, with a focus on secure authentication, role-based access, and data associations between users and service requests. I gained hands-on experience with session management, form validation, and prepared statements to prevent SQL injection. I also learned how to design a scalable database structure to track user-specific data like job history and appointments. Additionally, I deepened my understanding of how to maintain dynamic data relationships — ensuring users always see updated information without breaking associations. Overall, the project helped me strengthen both my frontend and backend development skills while reinforcing best practices in user experience, data integrity, and system design.



Saved: 8/5/2025 10:08:23 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:

Commit the changes and uploading to the server.



Saved: 8/5/2025 10:08:40 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:

Heroku was the hardest part because it was a long-going struggle so used the NJIT web server to publish the whole site.



Saved: 8/5/2025 10:08:16 PM