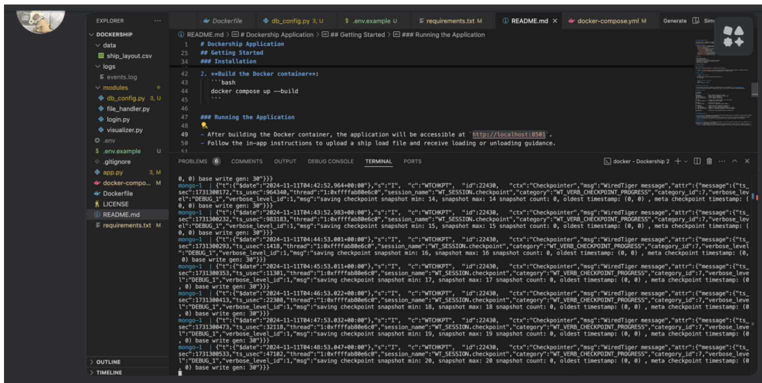
On November 24th, we identified the following test cases for our UI, Grid, Mongo Database connection system:

- **User Interface Improvements**: We tested the overall UI and identified areas needing improvement. Specifically, we worked on improving layout clarity, such as resizing the grid for better visibility and organizing input/output displays to reduce user confusion.
- **Grid functionality**: We found that the grid was not fully functional for container operations. We created test cases to ensure the grid allows loading, unloading, and balancing containers in real time without errors, while visually reflecting all changes accurately.
- **Fixing button operations**: Several buttons, such as “Load,” “Unload,” and “Balance”, did not respond as intended. We tested fixes to ensure each button performs its designated function, updates the grid state, and provides feedback to the user (e.g., success or error messages).
- **Real-time grid updates**: We tested scenarios where container movements (load/unload/balance) were not reflected immediately on the grid. We ensured that all operations now update the grid in real time to improve user experience and accuracy.
- **Button responsiveness**: We observed that some buttons were either unresponsive or required multiple clicks to trigger. Test cases confirmed that buttons are now functional, with single-click operations and no lag in response time.
- **Error messages for invalid inputs**: We identified a lack of error handling for incorrect or missing inputs. We implemented and tested error messages to inform users of invalid input formats, ensuring a smoother workflow.
- **Connecting to MongoDB database**: We tested the system’s connection to a MongoDB database to ensure that all container data, including names, weights, and positions, is stored and retrieved correctly. We verified that updates to the grid and manifest reflect in the database in real time.



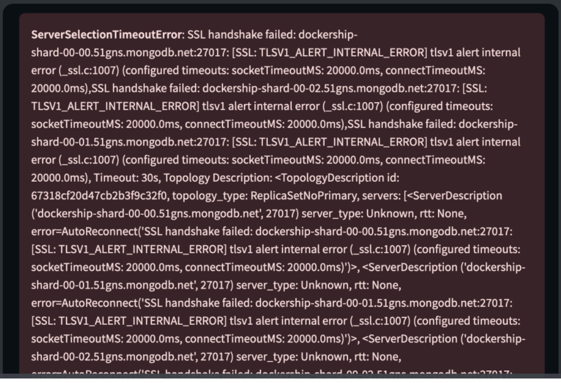
~~~
README.md > # Dockership Application > ## Getting Started > ### Running the Application
# Dockership Application
## Getting Started
### Installation
2. **Build the Docker container**:
   ```bash
   docker compose up --build
   ```

### Running the Application

- After building the Docker container, the application will be accessible at `http://localhost:8501`.
- Follow the in-app instructions to upload a ship load file and receive loading or unloading guidance.
~~~

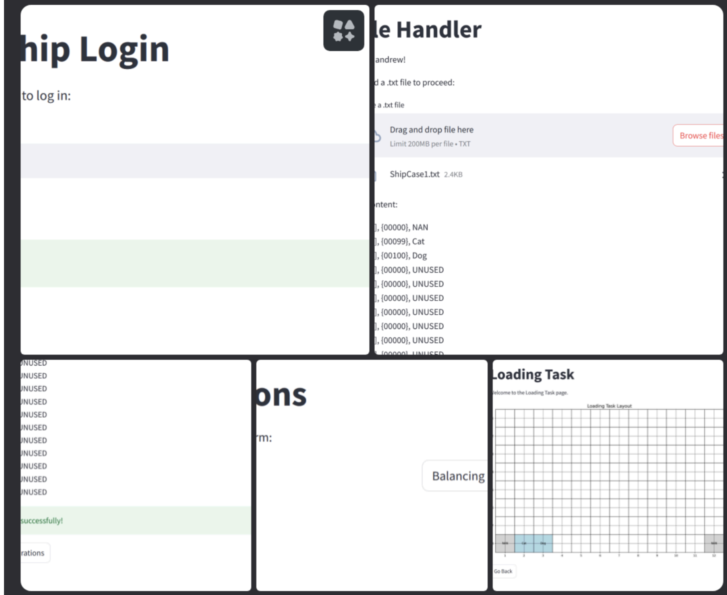
```
0, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:42:52.964+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300172,"ts_usec":964340,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_lev
el":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 14, snapshot max: 14 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (
0, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:43:52.983+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300232,"ts_usec":983183,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_lev
el":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 15, snapshot max: 15 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (
0, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:44:53.001+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300293,"ts_usec":1418,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_level
":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 16, snapshot max: 16 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0,
 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:45:53.011+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300353,"ts_usec":11301,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_leve
l":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 17, snapshot max: 17 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0
, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:46:53.022+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300413,"ts_usec":22308,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_leve
l":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 18, snapshot max: 18 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0
, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:47:53.032+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300473,"ts_usec":32118,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_leve
l":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 19, snapshot max: 19 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0
, 0) base write gen: 30"}}}
mongo-1  | {"t":{"$date":"2024-11-11T04:48:53.047+00:00"},"s":"I",  "c":"WTCHKPT",  "id":22430,   "ctx":"Checkpointer","msg":"WiredTiger message","attr":{"message":{"ts_
sec":1731300533,"ts_usec":47102,"thread":"1:0xffffab80e6c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"verbose_leve
l":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 20, snapshot max: 20 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0
, 0) base write gen: 30"}}}
```

the terminal keeps hanging



ServerSelectionTimeoutError: SSL handshake failed: dockership-shard-00-00.51gns.mongodb.net:27017: [SSL: TLSV1_ALERT_INTERNAL_ERROR] tlsv1 alert internal error (_ssl.c:1007) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms),SSL handshake failed: dockership-shard-00-02.51gns.mongodb.net:27017: [SSL: TLSV1_ALERT_INTERNAL_ERROR] tlsv1 alert internal error (_ssl.c:1007) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms),SSL handshake failed: dockership-shard-00-01.51gns.mongodb.net:27017: [SSL: TLSV1_ALERT_INTERNAL_ERROR] tlsv1 alert internal error (_ssl.c:1007) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms), Timeout: 30s, Topology Description: <TopologyDescription id: 67318cf20d47cb2b3f9c32f0, topology_type: ReplicaSetNoPrimary, servers: [<ServerDescription ('dockership-shard-00-00.51gns.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('SSL handshake failed: dockership-shard-00-00.51gns.mongodb.net:27017: [SSL: TLSV1_ALERT_INTERNAL_ERROR] tlsv1 alert internal error (_ssl.c:1007) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms)')>, <ServerDescription ('dockership-shard-00-01.51gns.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('SSL handshake failed: dockership-shard-00-01.51gns.mongodb.net:27017: [SSL: TLSV1_ALERT_INTERNAL_ERROR] tlsv1 alert internal error (_ssl.c:1007) (configured timeouts: socketTimeoutMS: 20000.0ms, connectTimeoutMS: 20000.0ms)')>, <ServerDescription ('dockership-shard-00-02.51gns.mongodb.net', 27017) server_type: Unknown, rtt: None,

i made changes to the ui and made the grid functional, does someone else want to start working on actually implementing the search algorithms for the loading and balancing functions

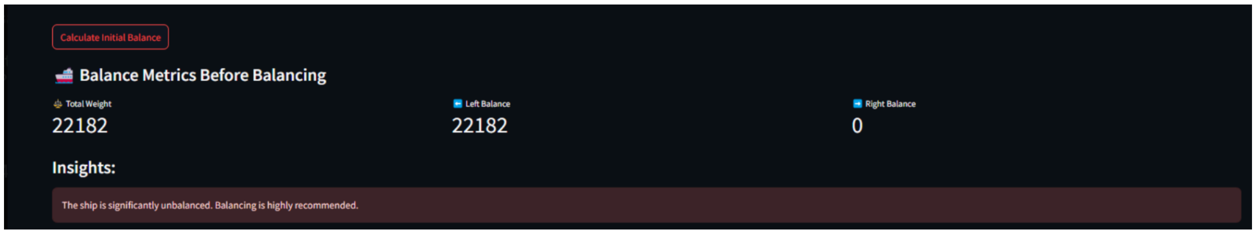


you might have to double click on some buttons for them to work!!

👍 1

On December 7th, we identified the following test cases for our ship container balancing and manifest downloading system:

- **Balancing accuracy:** We tested the system’s ability to detect an unbalanced ship and reposition containers to achieve balance. Test cases included both simple (few containers) and complex (multiple containers with varying weights) scenarios to ensure accuracy.
- **Time-efficient balancing:** We verified that the system minimizes the number of moves and time required (60 seconds per slot) to achieve balance, optimizing performance for larger container sets.
- **Manifest download functionality:** We tested the ability to generate and download the updated manifest after balancing operations. The downloaded manifest was checked to ensure it accurately reflects all container positions, names, and weights.

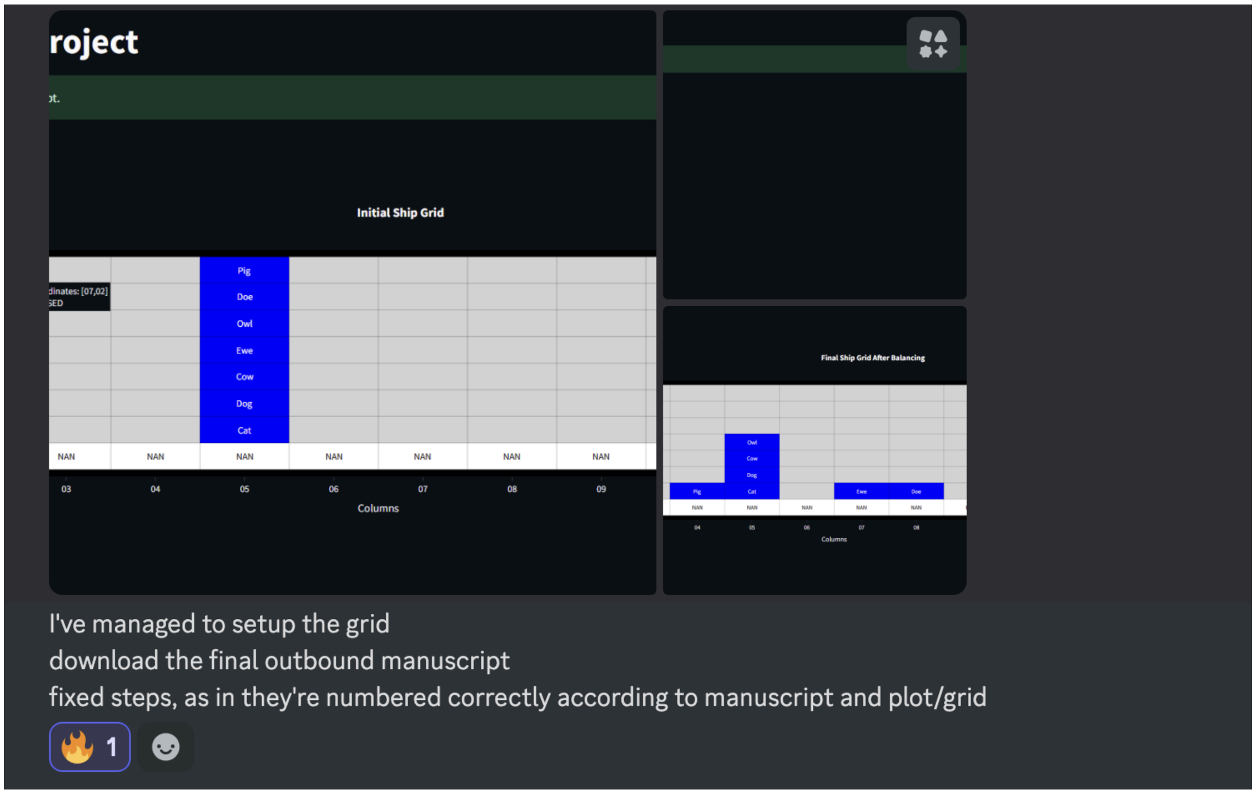

Calculate Initial Balance

Balance Metrics Before Balancing

Total Weight	Left Balance	Right Balance
22182	22182	0

Insights:

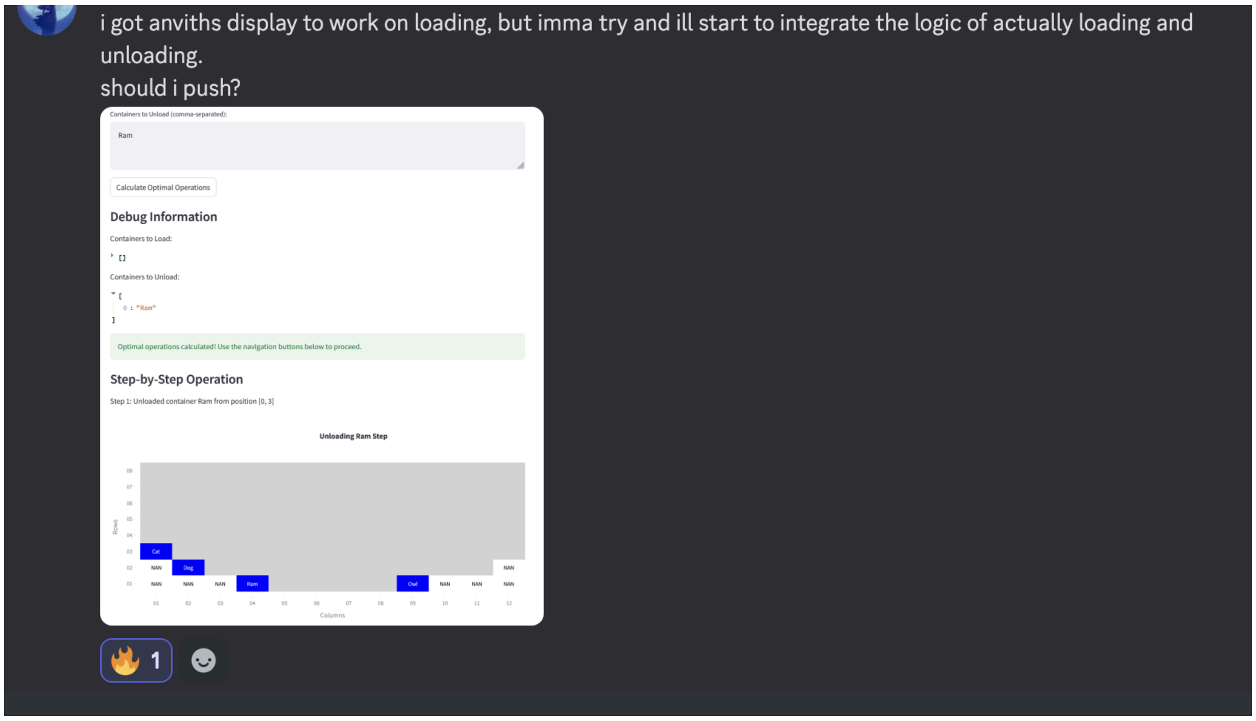
The ship is significantly unbalanced. Balancing is highly recommended.



I've managed to setup the grid
download the final outbound manuscript
fixed steps, as in they're numbered correctly according to manuscript and plot/grid

On December 9th, we identified the following test cases for our ship container loading/unloading system:

- **Grid upside down**: We observed that the grid displayed in reverse order, causing incorrect container positioning. We tested fixes to ensure the grid renders from top to bottom correctly.
- **Unloading logic error**: When unloading a container located underneath another container, the container above was left floating. We created test cases to confirm that the system prevents the removal of bottom containers until those above are unloaded.
- **Visualization of steps**: We found that the system does not display the balancing steps. To address this, we designed scenarios that ensure all moves are visualized step-by-step for improved clarity.
- **Loading new containers**: We discovered that loading new containers does not work as expected. We tested fixes to verify that new containers can be added, positioned and factored into the balancing logic correctly.
- **Displaying container weights**: We tested scenarios to ensure container weights are displayed properly, either by hovering over the container or directly on the grid. This allows users to know the weight inputs during balancing operations.
- **Input box prompt**: We identified the need for a clear prompt or example for container input. We tested solutions to guide users on input format (e.g., *"Container Name, Weight"*) to ensure proper container registration.



i got anviths display to work on loading, but imma try and ill start to integrate the logic of actually loading and unloading.
should i push?

Containers to Unload (comma-separated):

Ram

Debug Information

Containers to Load:

```
[]
```

Containers to Unload:

```
[
  0 : "Ram"
]
```

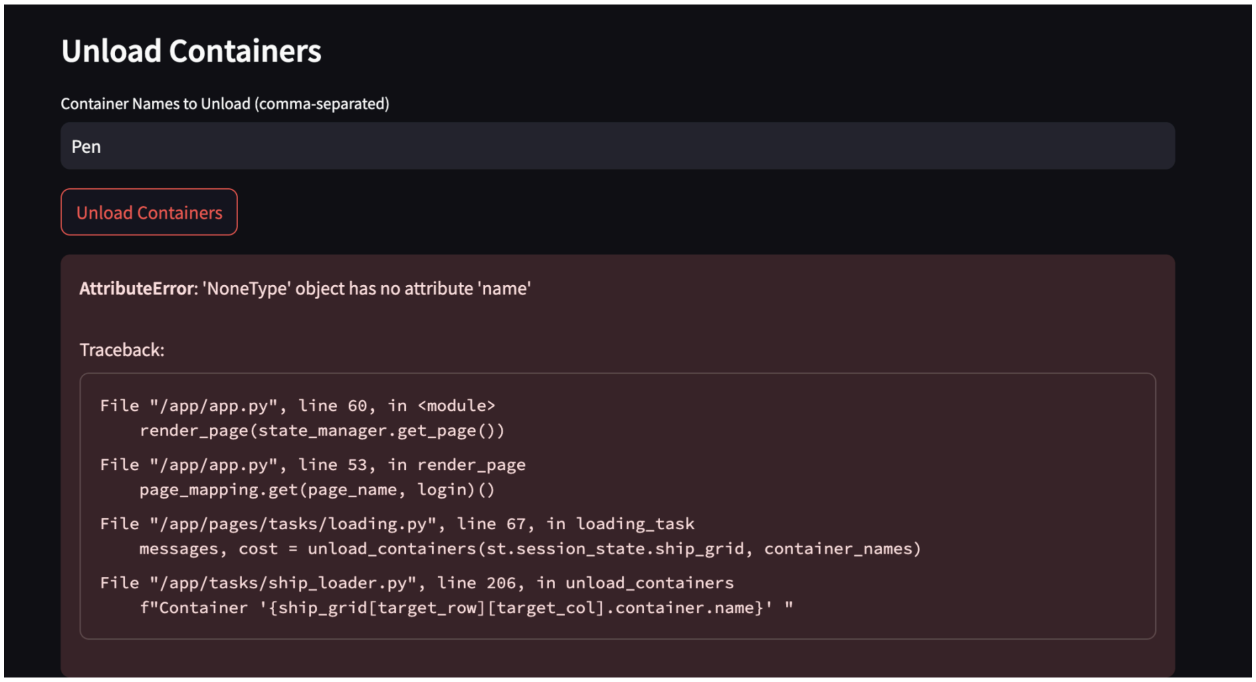
Optimal operations calculated! Use the navigation buttons below to proceed.

Step-by-Step Operation

Step 1: Unloaded container Ram from position [0, 3]

On December 12th, we identified the following test cases for our ship container loading and unloading + buffer system:

- **Input container list**: We tested the scenario where the user inputs a list of container names to be loaded or unloaded from the ship. This verifies that the system correctly processes and updates each container.
- **Unknown weight during load**: Since we do not know the container's weight until it's loaded, we ran tests to confirm that the system correctly assigns weight only upon successful placement on the grid.
- **Move cost optimization**: Each move across a slot costs 60 seconds. We created cases to minimize loading costs to evaluate loading and unloading sequences and their corresponding time efficiencies.
- **Origin position testing**: We checked that the first container to be loaded or unloaded starts at the top leftmost grid position. If the position is occupied during loading, the system correctly proceeds to the next available slot. Similarly, when unloading, the system locates the nearest container when no direct match exists.
- **Unique ID assignment**: We verified that each container receives a unique ID upon loading. We also tested updates to the manifest to ensure all loaded and unloaded containers reflect accurate information, as required by the balance files.
- **Time-efficient operations**: Finally, we designed scenarios to confirm the system loads and unloads containers in the most time-efficient manner possible, meeting the goal of minimizing overall move costs.



Unload Containers

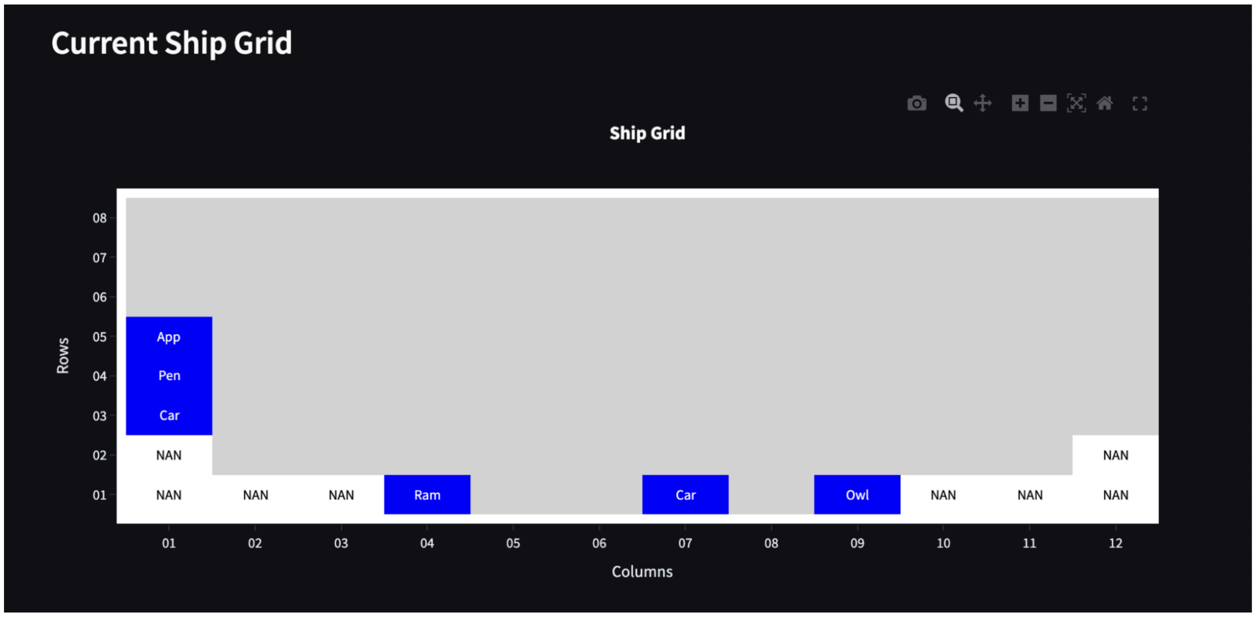
Container Names to Unload (comma-separated)

Pen

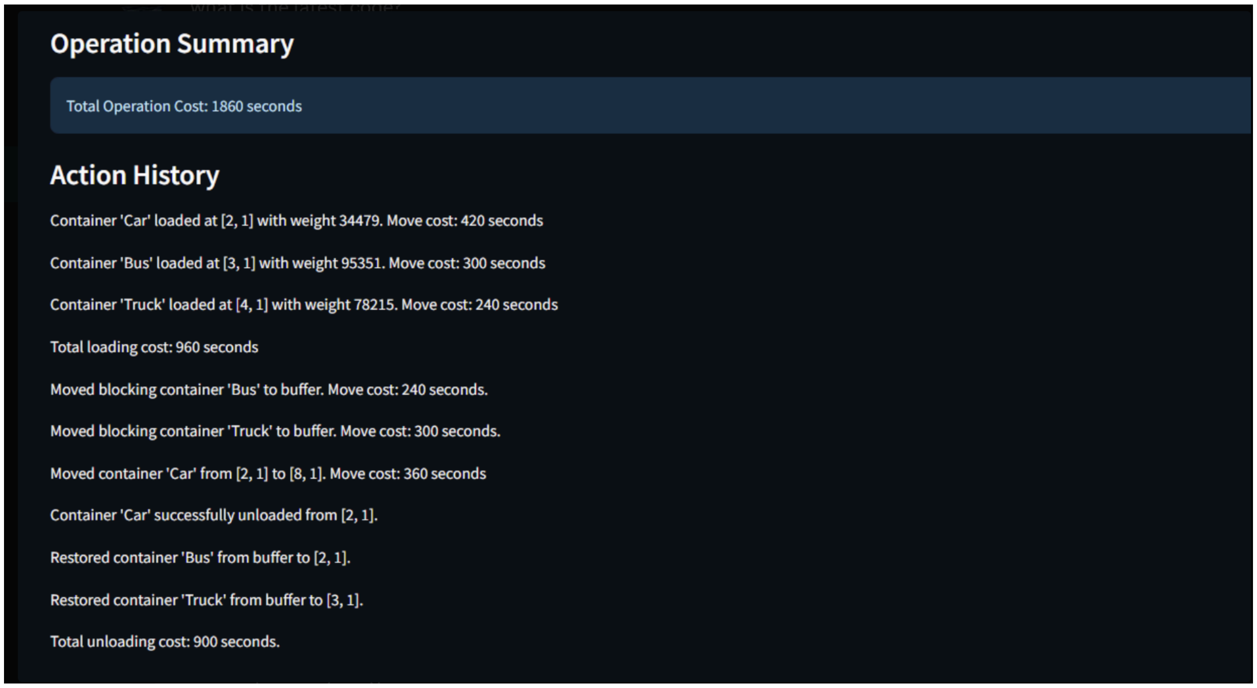
AttributeError: 'NoneType' object has no attribute 'name'

Traceback:

```
File "/app/app.py", line 60, in <module>
    render_page(state_manager.get_page())
File "/app/app.py", line 53, in render_page
    page_mapping.get(page_name, login)()
File "/app/pages/tasks/loading.py", line 67, in loading_task
    messages, cost = unload_containers(st.session_state.ship_grid, container_names)
File "/app/tasks/ship_loader.py", line 206, in unload_containers
    f"Container '{ship_grid[target_row][target_col].container.name}' "
```



Current Ship Grid



Operation Summary

Total Operation Cost: 1860 seconds

Action History

Container 'Car' loaded at [2, 1] with weight 34479. Move cost: 420 seconds

Container 'Bus' loaded at [3, 1] with weight 95351. Move cost: 300 seconds

Container 'Truck' loaded at [4, 1] with weight 78215. Move cost: 240 seconds

Total loading cost: 960 seconds

Moved blocking container 'Bus' to buffer. Move cost: 240 seconds.

Moved blocking container 'Truck' to buffer. Move cost: 300 seconds.

Moved container 'Car' from [2, 1] to [8, 1]. Move cost: 360 seconds

Container 'Car' successfully unloaded from [2, 1].

Restored container 'Bus' from buffer to [2, 1].

Restored container 'Truck' from buffer to [3, 1].

Total unloading cost: 900 seconds.

On December 15th, we identified the following additional test cases for ship container operations:

- **Max and min weight capacity:** We tested scenarios to ensure the ship does not exceed its maximum weight capacity and handles minimum weight requirements appropriately.
- **Unload non-existent containers**: We created cases where users attempt to unload containers not present on the ship, verifying that the system correctly rejects such requests and provides meaningful feedback.
- **Load to a full ship capacity:** We tested scenarios to confirm the system handles loading containers until the ship reaches full capacity, ensuring accurate updates to the manifest and grid.
- **Buffer implementation when ship is over 50% capacity**: We designed test cases to validate the buffer system activates correctly when the ship's capacity exceeds 50%, optimizing balance operations.
- **Handle container of origin:** We tested scenarios to ensure the system tracks and manages the origin of each container during loading, unloading, and balancing.
- **Handle list of multiple containers**: We verified that the system processes a list of multiple containers efficiently, updating the manifest and grid accurately.
- **Correctly updates and downloads manifest from either the loading/unloading or balance operation**: We tested that the manifest is updated and downloadable after every operation, reflecting the current state of the ship.
- **Balance a full ship:** We designed test cases to confirm the system effectively balances a fully loaded ship without errors.
- **Sequence of loading and unloading:** We evaluated the system's ability to handle sequential operations, ensuring no errors arise from complex sequences.
- **Sequence of balance**: Finally, we tested the system's ability to handle multiple balance operations in sequence, verifying stability and accuracy throughout.