I had a lot of fun with this project. Given that this was the first time I've coded in React / Typescript, and that I only had 48 hours, I set some specific and tiered goals inspired by the Acceptance Criteria.

Goals

Minimum viable product:

1. Have a button that adds a selected group of companies to "Liked Companies List".
2. Have a button that removes a selected group of companies from "Liked Companies List".
3. Have a button that selects all companies within the currently selected collection.
4. Have a button that deselects all selected companies.

Middle tier goals:

1. Have a button that adds a selected group of companies to any specified collection.
2. Have a button that removes a selected group of companies from the currently selected collection.
3. UX should reflect In Progress and Completed states when a lengthy action is being performed.
4. The user experience should be reasonably polished. (Details are subjective).

Reach goals:

1. Add in functionality to add or delete lists.
2. The user experience should be well polished. (Details are subjective).

Assumptions

Before diving into my approach, I want to talk about some basic assumptions.

1. **Performance**: I assumed that the operations would often involve large datasets (thousands of companies).
2. **User Experience**: Users need real-time feedback for long-running operations.
3. **Error Handling**: Some companies that I insert from one list into another list may already be associated with that other list. Attempting to insert duplicates may throw errors.

Approach

Since a lot of this was new to me, I started by building basic functionalities, and then learning from that experience to try expanding into harder or more involved ones.

The main deliverable was to allow users to add companies to “Liked Companies List” and “My List”, so I built two buttons for those. It made sense to also include functionality for removing companies from the currently selected collection.

Afterwards, the next step would be including a button that selects all companies in a collection, since the checkbox at the top of the UI only selects all companies displayed on the page. When it came to implementing “Select All”, I noticed that the checkboxes next to each individual company row were not checked, even though I had actually selected every company; this needed to be addressed as the UI should accurately reflect which companies are selected.

Adding thousands of companies to a list, however, posed a problem. This bulk operation took a long time, and if I wanted to implement a progress bar it wouldn’t work if I had to wait for the entire bulk operation to complete before returning a status. Therefore, I knew I had to implement some way of batching, and some way of displaying real-time information with that batching process. I looked up two options: **polling** and **websockets.** In the end I went with websockets.

Tradeoff 1: Polling vs. Websockets

As far as I understood, polling required periodic requests from the client to understand how much of the task had been completed, whereas websockets were more complicated to set up, but allowed two-way communication so that the server could initiate a response and send it to the client without the client asking. This was a pretty big consideration for me; my computer was already struggling to run tons of database operations with a bulk task, and adding more traffic didn’t sound appealing.

There was also the question of “when” to poll. If I was adding 1000 companies, would I poll every half-second? What if the entire operation was done in a half-second? Websockets didn’t have this problem as they could return progress updates with every “batch”, so if two users had different amounts of computational power with their clients, they would still receive similar experiences with the progress bar (the same granularity, just with different speeds).

End Tradeoff 1

This wasn’t really a tradeoff, but I also had to decide on when to show this websocket progress bar when performing an action. How would I know if an action was going to be really quick (and showing a progress bar may be startling to the user, and/or unnecessary?). For this I just used the size of the selection as a brief measure; if the number of companies selected was greater than the size of a batch, if I had to divide that task up, then I’d show the progress bar.

At some point I started running into problems with adding bulk companies to a list; when that collection already contained some companies in my selection, the duplicate company would cause the entire bulk change to fail when it couldn’t be inserted into the new list. This made me consider how I wanted to handle errors.

Tradeoff 2: Handling Errors

Should errors be visible to the user? I didn't think so.

On one hand, if errors were more visible it would be easier to see problems before they come up (for example, if a company failed to be added, the user may still assume that the company was added to the list).

On the other hand, some errors may happen very regularly, or even be **expected** to happen, and having those errors be visible would degrade the user experience.

In the end I decided to handle errors gracefully by making a note in the console, but not throwing an exception or stopping the long-running process, in case a company in that batch could not be inserted into a new list (for duplicate reasons).

End Tradeoff 2

User Experience Discussion

There were a few pain points in the starter UI that I wanted to fix.

1. It was hard to tell when an action was completed, as selected rows would remain selected.
    a. Selecting rows and performing an action would deselect those rows after the action is completed.
    b. Switching between collections would deselect any selected rows.
2. After making "select all" and "deselect all" buttons, the checkboxes on the UI that indicated whether a specific company was in the selection would not change.
    a. Select all and deselect all buttons also modify the checkboxes to show which companies are being selected and unselected.
3. Performing an action did not guarantee fresh data on the UI.
    a. After an action is performed, another call to fetch collection data for the collection in view is made. If i remove 5 items from a list, those 5 items are gone without needing to refresh the page.
4. When I added a progress bar, it wouldn't go away after it was completed.
    a. The act of switching to another collection would hide the progress bar and set it back to 0.

Some other considerations I implemented:

1. "Adding" buttons are disabled until some companies have been selected.
2. Automatic data refresh after completing bulk changes.

There are still quite a bit more steps to take to make the UI delightful to use to the customer.

1. Grouping buttons with the same functionality together, maybe into a "dropdown" button.
2. Turning "select all" and "deselect all" into the same button, where the button acts as a switch.

3. Allow creating and deleting new collections.
4. Making the progress bar persist (but hidden) when switching collections while performing a long-running task. Switching back to the original collection should bring the progress bar back.
5. Ensuring that companies are displayed in order of IDs in each collection (I believe they are in order for each page, but not globally for each collection).
6. This is more subjective, but perhaps a scrolling function that displays more companies as you get to the end of a page, than a strict page system.

Next Steps

1. Add a stop button to stop an in progress task if it's taking too long.
2. Clean up the code; there was a lot of repeated code, especially in the frontend. Some backend functions can likely be combined into one and accept a parameter (for example, adding and removing a company).
3. I think sometimes the UI would display companies in order of ID for that page, but not necessarily in order of the entire collection itself (for example, page 1 would be in order, but there may be a companyID on page 3 that should have come before the last entry on page 1). Fix this.
4. Create a dropdown option to select a list to “add selected companies” to, instead of having separate buttons for “My List” and “Liked Companies”. This would also support adding and deleting lists.
5. Made a change to one of the backend functions that I didn’t write, I think it was “fetch_companies_with_liked”. An error showed in my IDE about how there was a cartesian product in the SELECT statement; I did a quick fix to get this error message to leave but didn’t really understand it. Look into the nature of this problem.
6. Add functionality to add or drop lists.
7. Maybe instead of setting a constant batch size, look into finding this number dynamically depending on server load.
8. Better error reporting for the user; instead of interrupting the task or not reporting any errors at all, could find a happy medium where at the end of the task a summary of errors is given.
9. I know the database is intentionally throttled, but there can likely be some improvements to the way companies are added to a collection. Perhaps some way of joining tables in SQL.
10. More UI improvements.