Flow of data:

backend routes

```python
@history_routes.route('/')
@login_required
def get_history_entries():
    """Get all of the history entries."""
    entries = History.query.filter(History.user_id == current_user.id).order_by(History.updated_at.desc()).all()
    return {
        'history': [ entry.to_dict() for entry in entries ]
    }
```

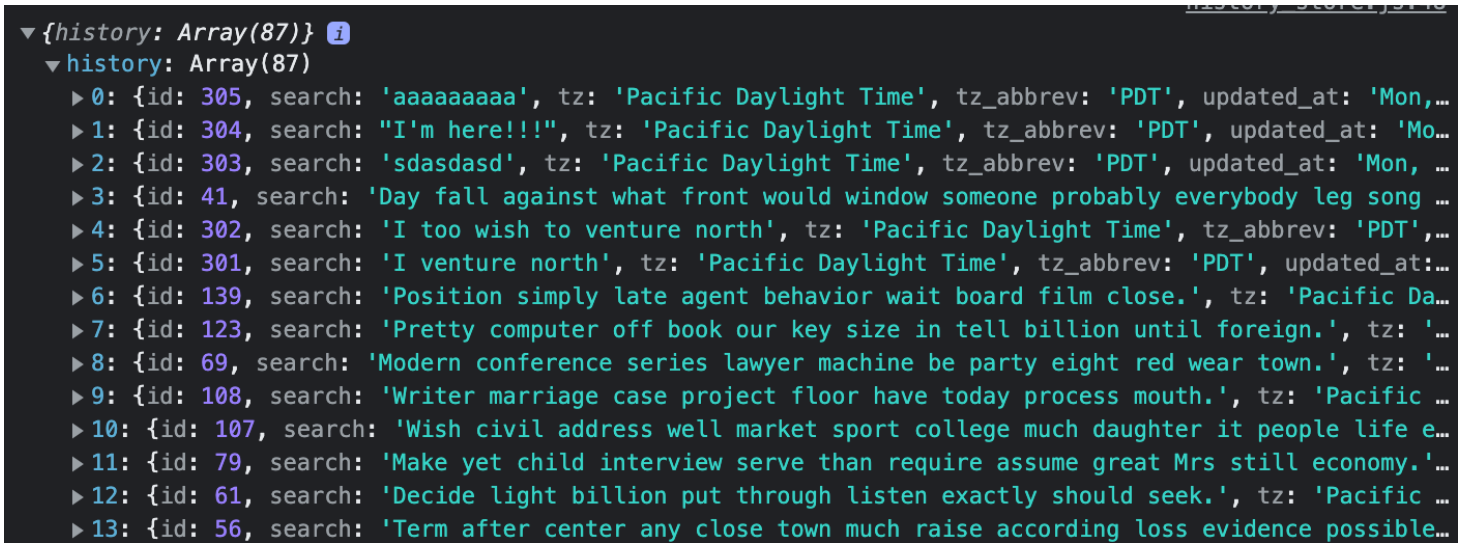
At this point, my object is this in the backend:

```
[<History 305>, <History 304>, <History 303>, <History 41>, <History 302>, <History 301>, <History 139>, <H
<History 103>, <History 141>, <History 182>, <History 143>, <History 179>, <History 202>, <History 255>, <H
, <History 170>, <History 154>, <History 268>, <History 197>, <History 295>, <History 198>, <History 195>,
, <History 216>, <History 274>, <History 298>, <History 125>, <History 250>, <History 269>, <History 90>, <
3>, <History 164>, <History 84>, <History 239>, <History 172>, <History 155>, <History 231>, <History 228>,
8>, <History 148>, <History 233>, <History 43>, <History 283>, <History 87>, <History 175>, <History 162>]
```

It's a list of 87 entries ordered how I would like. It is then parsed by json on the frontend here:

```javascript
export const updateHistoryEntry = (entry) => async dispatch => {
    const response = await fetch(`/creepycrawler/history/${entry.entryID}`, {
        headers: {
            'Content-Type': 'application/json'
        },
        method: 'PATCH',
        body: JSON.stringify({
            updated_at: entry.updated_at
        })
    })
    if (response.ok) {
        const entry = await response.json();
        dispatch(updateHistory(entry));
        return entry;
    }
}
```

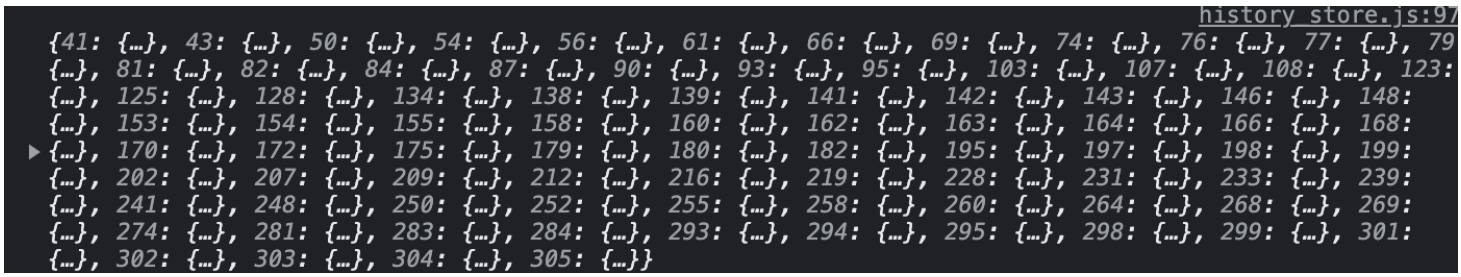
The order and functional structure is maintained, the only visible difference being that the indices are visible in the frontend console:



Where the data finally loses its order is in the READ_HISTORY case of my history reducer:

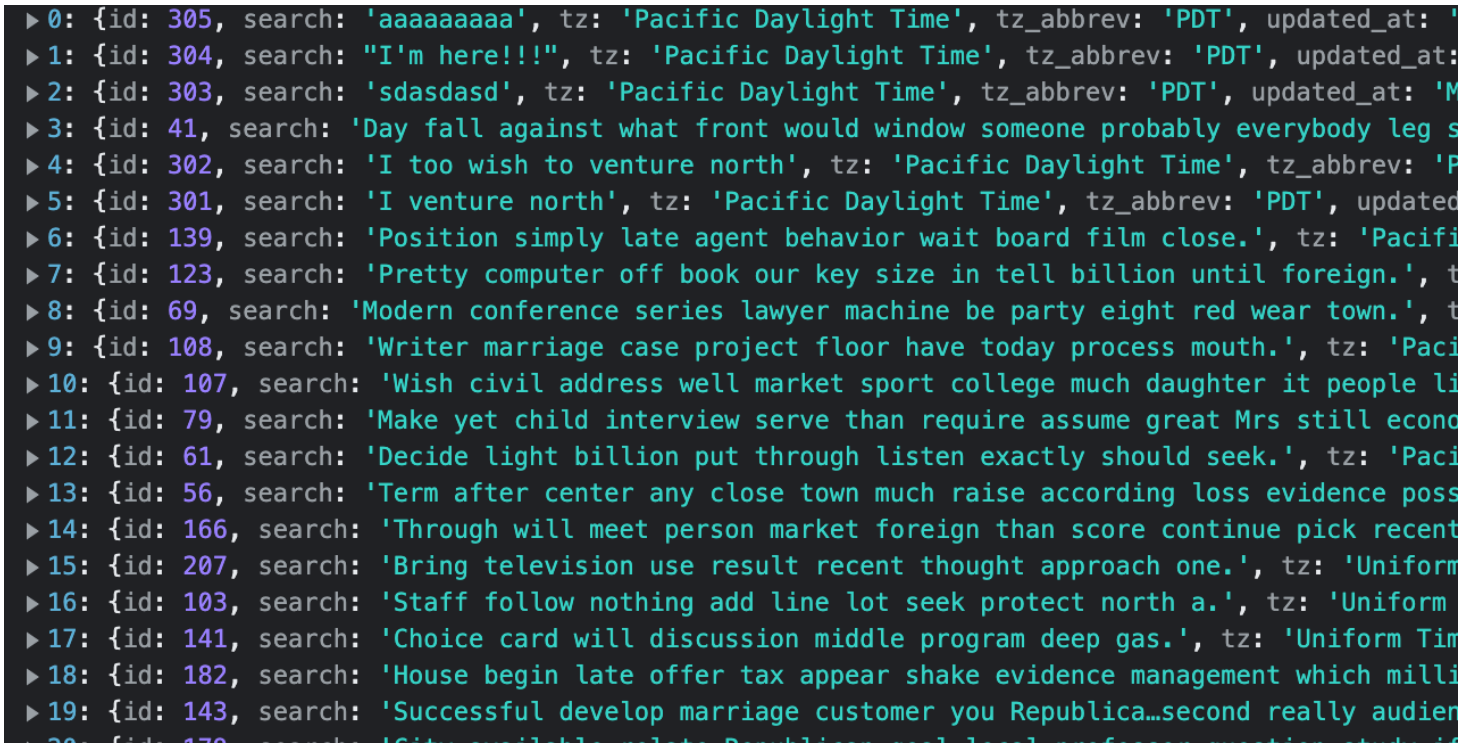
```
case READ_HISTORY:
    const entries = action.payload.history;
    entries.forEach(entry => newState[entry.id] = entry)
    console.log(newState);
```

The data is now ordered by index:



It will also appear as such in my state.

The conundrum is the following:
I need to have my indices match the ids for the sake of O(1) lookup time for my patch and delete operations. So, albeit I have been able to maintain the order of the data by spreading entries and newState in my READ_HISTORY case: `return {...entries,...newState}` instead of using a forEach method, the result is only beneficial for my read operation. As you'll see in the following picture, the re-indexing kills any instant lookup time chances:



I want instant lookup time as well as the order coming from my backend. Preferably, I would like to maintain this order rather than recreate it on the other side, through some frontend sort; however, I'm open to anything that allows me to attain this functionality without dramatically hurting the performance of my other operations.