

Flow of data:

backend routes

```
@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:

```
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:

```

▼ {history: Array(87)} ⓘ
  ▼ history: Array(87)
    ▶ 0: {id: 305, search: 'aaaaaaaa', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at: 'Mon,...
    ▶ 1: {id: 304, search: 'I'm here!!!', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at: 'Mo...
    ▶ 2: {id: 303, search: 'sdsdasd', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at: 'Mon, ...
    ▶ 3: {id: 41, search: 'Day fall against what front would window someone probably everybody leg song ...
    ▶ 4: {id: 302, search: 'I too wish to venture north', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT',...
    ▶ 5: {id: 301, search: 'I venture north', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at:...
    ▶ 6: {id: 139, search: 'Position simply late agent behavior wait board film close.', tz: 'Pacific Da...
    ▶ 7: {id: 123, search: 'Pretty computer off book our key size in tell billion until foreign.', tz: '...'
    ▶ 8: {id: 69, search: 'Modern conference series lawyer machine be party eight red wear town.', tz: '...'
    ▶ 9: {id: 108, search: 'Writer marriage case project floor have today process mouth.', tz: 'Pacific ...
    ▶ 10: {id: 107, search: 'Wish civil address well market sport college much daughter it people life e...
    ▶ 11: {id: 79, search: 'Make yet child interview serve than require assume great Mrs still economy.'...
    ▶ 12: {id: 61, search: 'Decide light billion put through listen exactly should seek.', tz: 'Pacific ...
    ▶ 13: {id: 56, search: 'Term after center any close town much raise according loss evidence possible...

```

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:

```

history_store.js:97
{41: {...}, 43: {...}, 50: {...}, 54: {...}, 56: {...}, 61: {...}, 66: {...}, 69: {...}, 74: {...}, 76: {...}, 77: {...}, 79: {...}, 81: {...}, 82: {...}, 84: {...}, 87: {...}, 90: {...}, 93: {...}, 95: {...}, 103: {...}, 107: {...}, 108: {...}, 123: {...}, 125: {...}, 128: {...}, 134: {...}, 138: {...}, 139: {...}, 141: {...}, 142: {...}, 143: {...}, 146: {...}, 148: {...}, 153: {...}, 154: {...}, 155: {...}, 158: {...}, 160: {...}, 162: {...}, 163: {...}, 164: {...}, 166: {...}, 168: {...}, 170: {...}, 172: {...}, 175: {...}, 179: {...}, 180: {...}, 182: {...}, 195: {...}, 197: {...}, 198: {...}, 199: {...}, 202: {...}, 207: {...}, 209: {...}, 212: {...}, 216: {...}, 219: {...}, 228: {...}, 231: {...}, 233: {...}, 239: {...}, 241: {...}, 248: {...}, 250: {...}, 252: {...}, 255: {...}, 258: {...}, 260: {...}, 264: {...}, 268: {...}, 269: {...}, 274: {...}, 281: {...}, 283: {...}, 284: {...}, 293: {...}, 294: {...}, 295: {...}, 298: {...}, 299: {...}, 301: {...}, 302: {...}, 303: {...}, 304: {...}, 305: {...}}

```

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:

```
▶0: {id: 305, search: 'aaaaaaaa', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at: '
▶1: {id: 304, search: 'I'm here!!!', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at:
▶2: {id: 303, search: 'sdasdasd', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated_at: 'M
▶3: {id: 41, search: 'Day fall against what front would window someone probably everybody leg s
▶4: {id: 302, search: 'I too wish to venture north', tz: 'Pacific Daylight Time', tz_abbrev: 'P
▶5: {id: 301, search: 'I venture north', tz: 'Pacific Daylight Time', tz_abbrev: 'PDT', updated
▶6: {id: 139, search: 'Position simply late agent behavior wait board film close.', tz: 'Pacifi
▶7: {id: 123, search: 'Pretty computer off book our key size in tell billion until foreign.', t
▶8: {id: 69, search: 'Modern conference series lawyer machine be party eight red wear town.', t
▶9: {id: 108, search: 'Writer marriage case project floor have today process mouth.', tz: 'Paci
▶10: {id: 107, search: 'Wish civil address well market sport college much daughter it people li
▶11: {id: 79, search: 'Make yet child interview serve than require assume great Mrs still econo
▶12: {id: 61, search: 'Decide light billion put through listen exactly should seek.', tz: 'Paci
▶13: {id: 56, search: 'Term after center any close town much raise according loss evidence poss
▶14: {id: 166, search: 'Through will meet person market foreign than score continue pick recent
▶15: {id: 207, search: 'Bring television use result recent thought approach one.', tz: 'Uniform
▶16: {id: 103, search: 'Staff follow nothing add line lot seek protect north a.', tz: 'Uniform
▶17: {id: 141, search: 'Choice card will discussion middle program deep gas.', tz: 'Uniform Tim
▶18: {id: 182, search: 'House begin late offer tax appear shake evidence management which milli
▶19: {id: 143, search: 'Successful develop marriage customer you Republica...second really audien
▶20: {id: 170, search: 'Citizens will be able to find a way to deal with the problem of the
```

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.