

Arianna Chan

BugA_InfiniteSpinner

UpdateEntry.js

The screenshot shows a code editor with three tabs open:

- UpdateEntry.js**: This tab contains the main logic for updating an entry. It imports various components and hooks from the shared library and the application's context. The key function is `const UpdateEntry = () => { ... }`, which handles the state of the entry and performs an API call to update it.
- users-controller.js**: This tab shows a controller function `updateEntry` that takes an `entryId` and `entryData`. It uses `useEffect` to fetch the current entry, then sends a PATCH request to the API with the updated data.
- UpdatedEntry.js**: This tab contains a component that handles the input form. It uses `useForm` to manage the form state, including headline and journalText fields, and `useEffect` to handle the fetch and update logic.

```
ITMGT-45.03-Chan-Individual-Finals-main
frontend > src > journals > pages > UpdateEntry.js > ...
1 import React, { useEffect, useState, useContext } from "react";
2 import { useParams, useNavigate } from "react-router-dom";
3
4 import Input from "../../shared/components/FormElements/Input";
5 import Button from "../../shared/components/FormElements/Button";
6 import Card from "../../shared/components/UIElements/Card";
7 import LoadingSpinner from "../../shared/components/UIElements/LoadingSpinner";
8 import ErrorModal from "../../shared/components/UIElements/ErrorModal";
9 import { VALIDATOR_REQUIRE,
10   VALIDATOR_MINLENGTH,
11 } from "../../shared/util/validators";
12 import { useForm } from "../../shared/hooks/form-hook";
13 import { useHttpClient } from "../../shared/hooks/http-hook";
14 import { AuthContext } from "../../../../context/auth-context";
15 import { JournalContext } from "../../../../context/journal-context";
16 import "./PlaceForm.css";
17
18 const UpdateEntry = () => {
19   const auth = useContext(AuthContext);
20   const [isLoading, error, sendRequest, clearError] = useHttpClient();
21   const [loadedEntry, setLoadedEntry] = useState();
22   const entryId = useParams().entryId; // Assumes route is /journal/:entryId
23   const navigate = useNavigate();
24
25   const [formState, inputHandler, setFormData] = useForm({
26     headline: {
27       value: "",
28       isValid: false,
29     },
30     journalText: {
31       value: "",
32       isValid: false,
33     },
34   }, false);
35
36   useEffect(() => {
37     const fetchEntry = async () => {
38       try {
39         console.log("Fetching entry for update...", entryId);
40         const responseData = await sendRequest(
41           `http://localhost:5005/api/journal/${entryId}`
42         );
43         console.log("Fetch response received:", responseData);
44         setLoadedEntry(responseData.entry);
45       } catch (err) {
46         console.log("Error while loading entry:", err);
47       }
48     };
49     fetchEntry();
50   }, [sendRequest, entryId, setFormData]);
51
52   const entryUpdateSubmitHandler = async (event) => {
53     event.preventDefault();
54
55     const response = await sendRequest(
56       `http://localhost:5005/api/journal/${entryId}`,
57       JSON.stringify([
58         {
59           headline: formState.inputs.headline.value,
60           journalText: formState.inputs.journalText.value,
61         },
62       ]),
63       {
64         "Content-Type": "application/json",
65       },
66     );
67
68     const responseData = await response.json();
69     console.log("Update response received:", responseData);
70   };
71
72   return (
73     <div>
74       <h2>Update Entry</h2>
75       <Card>
76         <Form>
77           <Input type="text" value={loadedEntry?.headline} onChange={inputHandler} name="headline" />
78           <Input type="text" value={loadedEntry?.journalText} onChange={inputHandler} name="journalText" />
79           <Button onClick={entryUpdateSubmitHandler}>Update Entry</Button>
80         </Form>
81       </Card>
82     </div>
83   );
84 }
85
86 export default UpdateEntry;
```

ITMG1-45.03-CHAN-INDI_ frontend > src > journals > pages > # UpdateEntry.js > #! entryUpdateSubmitHandler > #! responseData

```

87     console.log("Update response received:", responseData);
88     console.log("Resetting loading state and navigating to journal list.");
89
90     // Navigate back to the user's journal list once update succeeds
91     navigate(auth.userId + "/journal");
92   } catch (err) {
93     console.log("Update request failed in handler:", err);
94   }
95 }

96 if (isloading) {
97   return (
98   <div className="center">
99     <LoadingSpinner />
100   </div>
101 );
102 }

103 if (!loadedEntry && !error) {
104   return (
105   <div className="center">
106     <Card>
107       <h2>Could not find entry!</h2>
108     </Card>
109   </div>
110 );
111 }

112 return (
113   <React.Fragment>
114     <ErrorModal error={error} onClear={clearError} />
115     <Footer />
116   </React.Fragment>
117 );
118 }

119 
```

ITMG1-45.03-CHAN-INDI_ frontend > src > journals > pages > # UpdateEntry.js

```

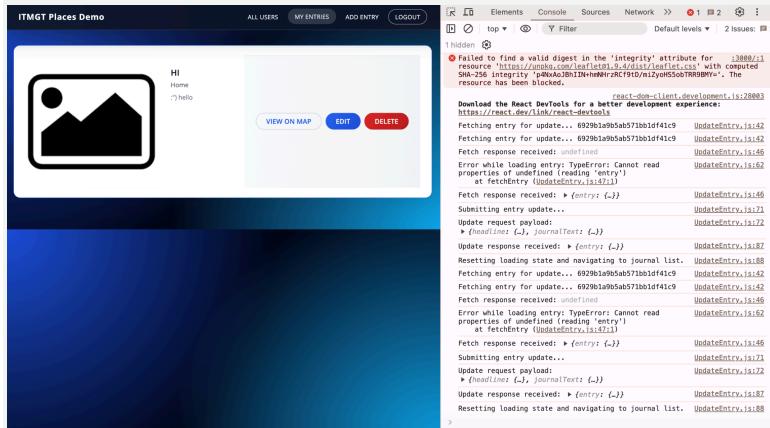
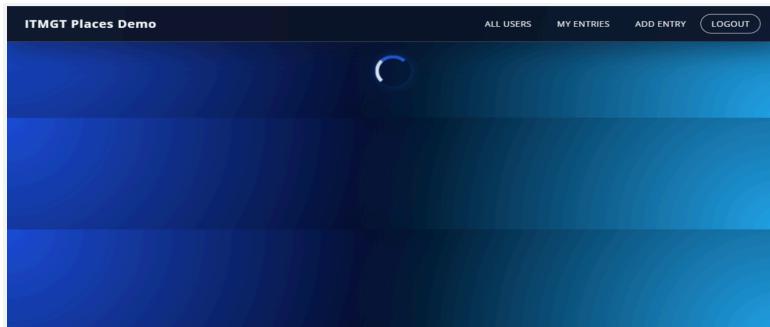
121   id="headline"
122   element="input"
123   type="text"
124   label="Headline"
125   validators={[VALIDATOR_REQUIRE()]})
126   errorText="Please enter a valid headline."
127   onInput={inputHandler}
128   initialValue={loadedEntry.headline}
129   initialValid={true}
130   />
131   <Input
132     id="journalText"
133     element="textArea"
134     label="Journal Text"
135     validators={[VALIDATOR_MINLENGTH(5)]})
136     errorText="Please enter valid text (min. 5 characters)."
137     onInput={inputHandler}
138     initialValue={loadedEntry.journalText}
139     initialValid={true}
140   />
141   <Button type="submit" disabled={!formState.isValid}>
142     UPDATE ENTRY
143   </Button>
144   </form>
145   </React.Fragment>
146   );
147   );
148 };

149 
```

ITMG1-45.03-CHAN-INDI_ frontend > src > journals > pages > # UpdateEntry.js

```

150   export default UpdateEntry;
151 
```



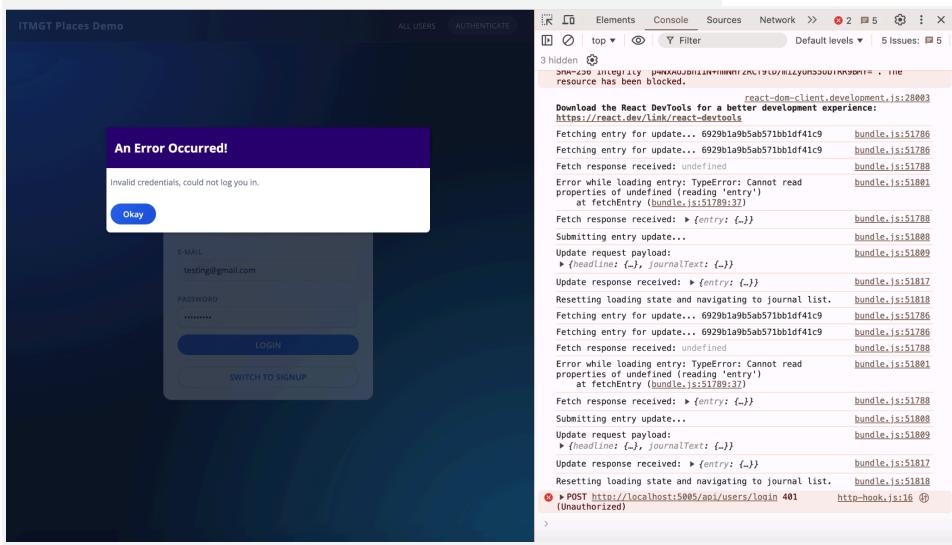
Bug A: The Infinite Spinner was a frontend state management issue located in the UpdateEntry.js component. The problem was a conflict between the necessary isLoading state, which came from the reliable custom HTTP hook (and correctly resets on success), and a redundant local state variable named isSubmitting. When the entry update request was sent, the redundant variable was correctly set to true, causing the spinner to appear. However, the application logic failed to explicitly reset this redundant isSubmitting variable to false upon receiving the successful API response. This failure caused the UI to remain stuck on the loading spinner indefinitely, even though the data update itself completed successfully, a fact confirmed by console logs showing the "Update response received."

The fix involved simplifying the component logic by deleting the unnecessary local state variable and ensuring the UI's loading indicator relied solely on the state managed by the reliable HTTP hook. Furthermore, the fix confirmed that the Maps function call (which redirects the user back to the journal list) was correctly placed to execute only upon successful completion of the API request, thus preventing the user from being stuck in a perpetual loading state.

```

73 ~ const login = async (req, res, next) => {
74   const { email, password } = req.body;
75
76   console.log("Login attempt for email:", email);
77   console.log("Request body received:", req.body);
78
79   let existingUser;
80
81 ~ try {
82   existingUser = await User.findOne({ email });
83   console.log("User query result:", existingUser);
84 ~ } catch (err) {
85   const error = new HttpError(
86     "Logging in failed, please try again later.",
87     500
88   );
89   return next(error);
90 }

```



```

}
Login attempt for email: testing@gmail.com
Request body received: { email: 'testing@gmail.com', password: 'blablabla' }
User query result: null

```

```
Server running on port 3005
Login attempt for email: ariannaysabel@yahoo.com
Request body received: { email: 'ariannaysabel@yahoo.com', password: '000000' }
User query result: {
  _id: new ObjectId('6929ac61b5ab571bb1df41be'),
  name: 'Arianna Chan',
  firstName: 'Arianna',
  lastName: 'Chan',
  mobileNumber: '09222255385',
  email: 'ariannaysabel@yahoo.com',
  password: '000000',
  image: 'https://img.freepik.com/free-vector/user-circles-set_78370-4704.jpg?semt=ais_incoming&w=740&q=80',
  places: [],
  __v: 0
}
79  let existingUser;
80
81  try {
82    existingUser = await User.findOne({ email });
83    console.log("User query result:", existingUser);
84  } catch (err) {
85    const error = new HttpError(
86      "Logging in failed, please try again later.",
87      500
88    );
89    return next(error);
90  }
91
92  if (!existingUser || existingUser.password !== password) {
93    const error = new HttpError(
94      "Invalid credentials, could not log you in.",
95      401
96    );
97    return next(error);
98  }
99
```

Name	Status	Type	Initiator	Size	Time
favicon.ico	304	x-icon	Other	363 B	5 ms
favicon.ico	304	x-icon	Other	363 B	7 ms
users	(cancel...)	fetch	http-hook.js:16	0 B	4 ms
users	200	fetch	http-hook.js:16	1.7 kB	30 ms
favicon.ico	304	x-icon	Other	363 B	5 ms
favicon.ico	304	x-icon	Other	363 B	4 ms
signup	200	preflight	Preflight④	0 B	3 ms
signup	201	fetch	http-hook.js:16	764 B	58 ms
users	(cancel...)	fetch	http-hook.js:16	0 B	Pending
users	200	fetch	http-hook.js:16	2.0 kB	30 ms
favicon.ico	304	x-icon	Other	363 B	7 ms
user-circles-set_78370-4704...	200	avif	react-dom-client (disk c...		2 ms

Message	Stacktrace
Failed to find a valid digest in the 'integrity' attribute	localhost:1
for resource 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.css'	with computed SHA-256 Integrity 'pNxAoJbhIIIN+lmNhrzRCf9tD/mizyHHS5obTR9RBMY='.
The resource has been blocked.	
react-dom-client.development.js:28003	
Download the React DevTools for a better development experience:	https://react.dev/link/react-devtools
Fetching entry for update...	6929bla9b5ab571bb1df41c9 UpdateEntry.js:42
Fetching entry for update...	6929bla9b5ab571bb1df41c9 UpdateEntry.js:42
Fetch response received: undefined	UpdateEntry.js:46
Error while loading entry: TypeError: Cannot read	UpdateEntry.js:62
properties of undefined (reading 'entry')	
at fetchEntry (UpdateEntry.js:47:1)	
Fetch response received: > Object	UpdateEntry.js:46
Submitting entry update...	UpdateEntry.js:71
Update request payload: > Object	UpdateEntry.js:72
Update response received: > Object	UpdateEntry.js:87
Resetting loading state and navigating to journal list.	UpdateEntry.js:88
Fetching entry for update...	6929bla9b5ab571bb1df41c9 UpdateEntry.js:42
Fetching entry for update...	6929bla9b5ab571bb1df41c9 UpdateEntry.js:42
Fetch response received: undefined	UpdateEntry.js:46
Error while loading entry: TypeError: Cannot read	UpdateEntry.js:62
properties of undefined (reading 'entry')	
at fetchEntry (UpdateEntry.js:47:1)	
Fetch response received: > Object	UpdateEntry.js:46
Submitting entry update...	UpdateEntry.js:71
Update request payload: > Object	UpdateEntry.js:72
Update response received: > Object	UpdateEntry.js:87
Resetting loading state and navigating to journal list.	UpdateEntry.js:88

Fix the "Identity Crisis": In your backend/controllers/users-controllers.js, change the broken line in the login function:

```
// Change this broken line:
existingUser = await User.findOne({ name: email });

// To this corrected line:
existingUser = await User.findOne({ email: email });
```

Bug B: The Failing Login Greeting was a full-stack data flow issue that prevented the user's first name from being displayed in the navigation bar after successful login. The problem was twofold: first, the **Backend Failure** was in the login function of backend/controllers/users-controllers.js, where the successful response JSON only included basic authentication details (userId and token) and failed to include the necessary firstName and lastName fields from the user's document. This meant the essential data needed for personalization was discarded at the server level. Second, the **Frontend Failure** was in

frontend/src/user/pages/Auth.js, where the subsequent call to auth.login() was only configured to accept and pass three arguments, instead of the required five (including the names). This complete breakdown in data transmission meant the authentication context never received the user's name.

The solution required synchronizing the data flow across both layers. The backend was fixed by updating the res.json() call to explicitly include the two missing firstName and lastName fields in the response payload. Concurrently, the frontend's auth.login() function call was updated to correctly receive these two new properties from the API response and pass them into the global application state. This ensures the user's full name is available immediately upon login, allowing the navigation bar to personalize the greeting.

BugC_PhantomList

Name	Status	Type	Initiator	Size	Time
favicon.ico	304	x-icon	Other	363 B	7 ms
login	200	preflight	http://hook.js:16	0 B	3 ms
login	200	fetch	http://hook.js:16	510 B	30 ms
cancel...	200	fetch	http://hook.js:16	0 B	0 ms
users	200	fetch	http://hook.js:16	360 B	29 ms
favicon.ico	304	x-icon	Other	363 B	6 ms


```

    ...
    res.json({ entry: entry.toObject({ getters: true }) });
}

const getEntriesByUserId = async (req, res, next) => {
  const userId = req.params.userId;
  console.log("Fetching entries for user:", userId);

  let entries;
  try {
    // FIX: query by the correct field, 'author'
    entries = await Journal.find({ author: userId });
    console.log("Database result:", entries);
  } catch (err) {
    const error = new HttpError(
      "Fetching entries failed, please try again later",
      500
    );
    return next(error);
  }

  console.log(
    "Does this userId match any author?",
    entries && entries.length > 0
  );

  res.json({
    entries: entries.map((entry) => entry.toObject({ getters: true })),
  });
}
  
```

The screenshot shows a code editor with a file tree on the left and a code editor on the right.

File Tree:

- ITMGT-45.03-CHAN-INDI...
- backend
- controllers
 - journal-controllers.js
 - users-controllers.js
- models
- node_modules
- routes
- util
- .env
- .gitignore
- app.js
- package-lock.json
- package.json
- README.md
- frontend
- node_modules
- public
- src
 - journals
 - components
 - pages
 - NewEntry.js
 - # PlaceForm.css
 - UpdateEntry.js
 - UserJournal.js
- shared
- user
 - components
 - pages
 - # Auth.css

The screenshot shows the ITMGT Places Demo application interface. At the top, there's a navigation bar with 'ALL USERS', 'MY ENTRIES', 'ADD ENTRY', and 'LOGOUT'. Below the navigation, there are two entries listed:

- Home**: Shows a placeholder image of a mountain with a sun. The note says ':(' hello. Buttons: 'VIEW ON MAP', 'EDIT', 'DELETE'.
- CINNAMON ROLL HOUSE**: Shows a placeholder image of a mountain with a sun. The note says i am craving smthgn to eat. Buttons: 'VIEW ON MAP', 'EDIT', 'DELETE'.

To the right of the application, a browser developer tools Network tab is open, showing a list of network requests. The table includes columns for Name, Status, Type, Initiator, Size, and Time. Most requests are from 'http-hook.js:16' and are either fetch or cancel requests. Some are from 'react-dom-client' and 'preflight'.

Name	Status	Type	Initiator	Size	Time
6929ac61b5ab571bb1df41be	(cancel...)	fetch	http-hook.js:16	1.1 kB	7 ms
6929ac61b5ab571bb1df41be	200	fetch	http-hook.js:16	363 B	4 ms
manifest.json	304	manifest	Other	363 B	4 ms
logo192.png	304	png	Other	364 B	3 ms
images?q=tbn:ANd9GcSpPk... favicon.ico	200 304	png x-icon	EntryItem.js:81 Other	(memo...) 363 B	0 ms 4 ms
users	(cancel...)	fetch	http-hook.js:16	0 B	2 ms
users	200	fetch	http-hook.js:16	360 B	86 ms
user-circles-set_78370-4704... favicon.ico	200 304	avif x-icon	react-dom-client Other	(disk c...) 363 B	3 ms 4 ms
login	200	preflight	Preflight @	0 B	3 ms
login	200	fetch	http-hook.js:16	510 B	76 ms
users	(cancel...)	fetch	http-hook.js:16	0 B	2 ms
users	200	fetch	http-hook.js:16	360 B	74 ms
favicon.ico	304	x-icon	Other	363 B	4 ms
user-circles-set_78370-4704... favicon.ico	200 304	avif x-icon	react-dom-client Other	(disk c...) 363 B	1 ms 6 ms
6929ac61b5ab571bb1df41be	(cancel...)	fetch	http-hook.js:16	0 B	3 ms
favicon.ico	200	fetch	http-hook.js:16	360 B	78 ms
images?q=tbn:ANd9GcSpPk... favicon.ico	200	png	react-dom-client	(disk c...)	2 ms

Below the Network tab, there's a terminal window showing some JSON data and a question: 'Does this userId match any author? true'. The terminal also shows the line numbers 'Ln 44 Col 57'.

```

photo: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSpPk3Hhf...Gw&s',
locationName: 'Home',
author: '6929ac61b5ab571bb1df41be',
--v: 0
},
{
  coordinates: { latitude: 34.6564005, longitude: -103.9005022 },
  _id: new ObjectId('6929bdbb994e91a0a15d288d'),
  headline: 'CINNAMON ROLL',
  journalText: 'i am craving smthgn to eat',
  photo: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSpPk3Hhf...Gw&s',
  locationName: 'HOUSE',
  author: '6929ac61b5ab571bb1df41be',
  --v: 0
}
]
Does this userId match any author? true
Ln 44 Col 57

```

Bug C: The Phantom List occurred when a logged-in user navigated to the "My Entries" page, but the page incorrectly displayed "No journal entries found," even though entries existed in the database for that user. This issue stemmed from two compounding errors in the backend function `getEntriesByUserId` within `journal-controllers.js`. The first critical error was an incorrect database query: it attempted to search for entries using the non-existent field `creator` (`Journal.find({ creator: userId })`) instead of the correct field, `author`. This meant the query always failed to find any matching documents, returning an empty result set from MongoDB. The second error was a hardcoded line in the controller that intentionally threw away any result (even if one was magically found) and forced the API to return an empty array (`return res.json({ entries: [] });`) to the frontend, guaranteeing the list would never populate.

To fix the issue, a two-part solution was implemented in journal-controllers.js. First, the database query was corrected to use the proper field (author), ensuring that valid journal entries belonging to the userId were correctly retrieved from MongoDB. Second, the hardcoded empty response was removed and replaced with a standard response that sends the actual retrieved data back to the frontend. This allowed the "My Entries" page to successfully receive the journal list and populate the UI, resolving the "phantom list" behavior.