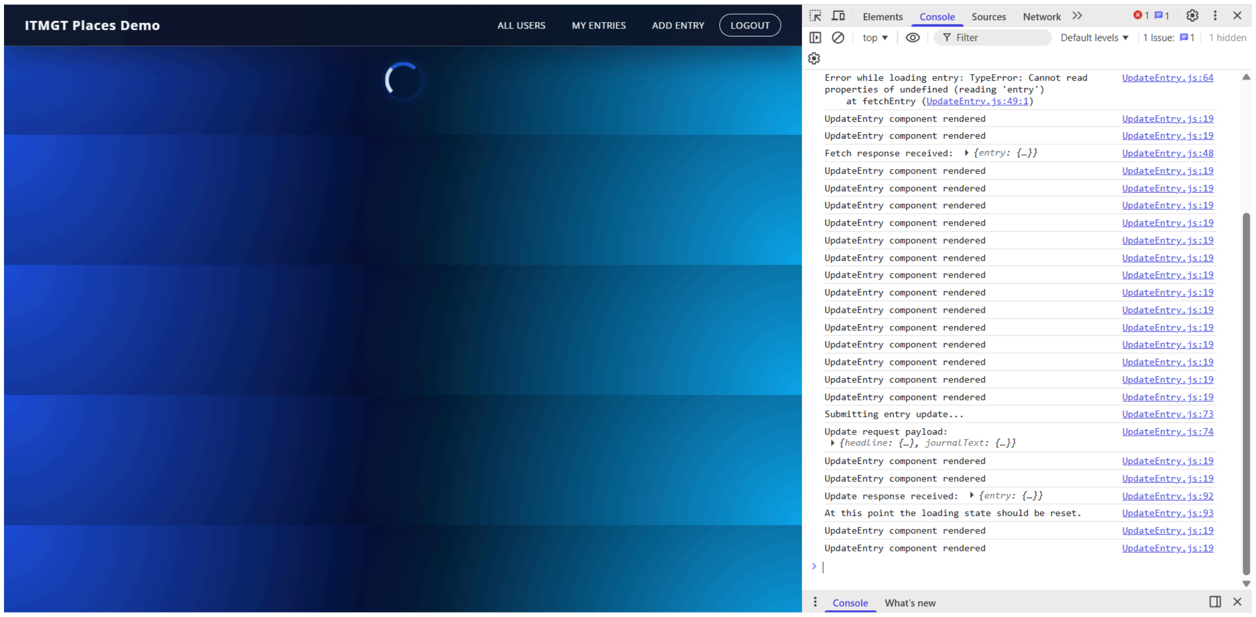
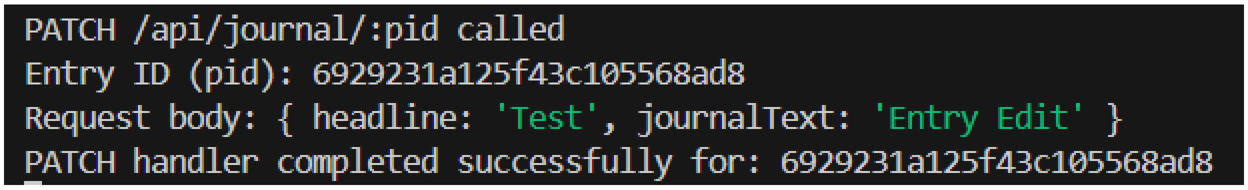
Ethan Gabriel F. Aquino 230408
3 BS ITE ITMGT 45.03: Debug Report

BUG A: The Infinite Spinner

Error Documentation:



```
Error while loading entry: TypeError: Cannot read            UpdateEntry.js:64
properties of undefined (reading 'entry')
    at fetchEntry (UpdateEntry.js:49:1)
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
Fetch response received:  ▸ {entry: {…}}                     UpdateEntry.js:48
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
Submitting entry update...                                   UpdateEntry.js:73
Update request payload:                                      UpdateEntry.js:74
 ▸ {headline: {…}, journalText: {…}}
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
Update response received:  ▸ {entry: {…}}                    UpdateEntry.js:92
At this point the loading state should be reset.             UpdateEntry.js:93
UpdateEntry component rendered                               UpdateEntry.js:19
UpdateEntry component rendered                               UpdateEntry.js:19
```



```
PATCH /api/journal/:pid called
Entry ID (pid): 6929231a125f43c105568ad8
Request body: { headline: 'Test', journalText: 'Entry Edit' }
PATCH handler completed successfully for: 6929231a125f43c105568ad8
```

After updating an entry, the PATCH request completes successfully (confirmed by backend terminal logs and frontend console output), but because the local `isSubmitting` *flag is never reset and the post-update navigation back to the user's journal list is skipped, the UI remains stuck on the loading spinner instead of returning to the updated journal view.*

Console Logs (Added to *entryUpdateSubmitHandler*):

```
const entryUpdateSubmitHandler = async (event) => {
  event.preventDefault();
  console.log("Submitting entry update...");
  console.log("Update request payload:", formState.inputs);
  try {
    const responseData = await sendRequest(
      `http://localhost:5005/api/journal/${entryId}`,
      "PATCH",
      JSON.stringify({
        headline: formState.inputs.headline.value,
        journalText: formState.inputs.journalText.value,
      }),
      {
        "Content-Type": "application/json",
      }
    );
    console.log("Update response received:", responseData);
    console.log("Resetting loading state and navigating to journal
list.");
    navigate("/" + auth.userId + "/journal");
  } catch (err) {
    console.log("Error caught:", err);
  }
};
```

These logs showed that the PATCH request completed successfully in both the browser console and Node terminal while the UI stayed on the spinner

Error 1: Extra submitting state keeps spinner active

```
const [isSubmitting, setIsSubmitting] = useState(false);
// ...
setIsSubmitting(true);
// ...
if (isLoading || isSubmitting) {
  return (
    <div className="center">
      <LoadingSpinner />
    </div>
  );
}
```

`isSubmitting` is set to `true` on submit and never reset, so the condition `isLoading ||` `isSubmitting` is always true after the first update, leaving the screen stuck on the loading spinner even though the request has finished.

Error 2: Missing navigation after successful update

```
// navigate("/" + auth.userId + "/journal");
```

Navigation back to the journal list was commented out, so even after a successful PATCH the user remained on the update page with the spinner instead of seeing the updated list.

Solution 1: Remove unnecessary submitting state and rely on isLoading

```
- const [isSubmitting, setIsSubmitting] = useState(false);
...
- setIsSubmitting(true);
...
- if (isLoading || isSubmitting) {
```

```
+ if (isLoading) {
    return (
      <div className="center">
        <LoadingSpinner />
      </div>
    );
  }
```

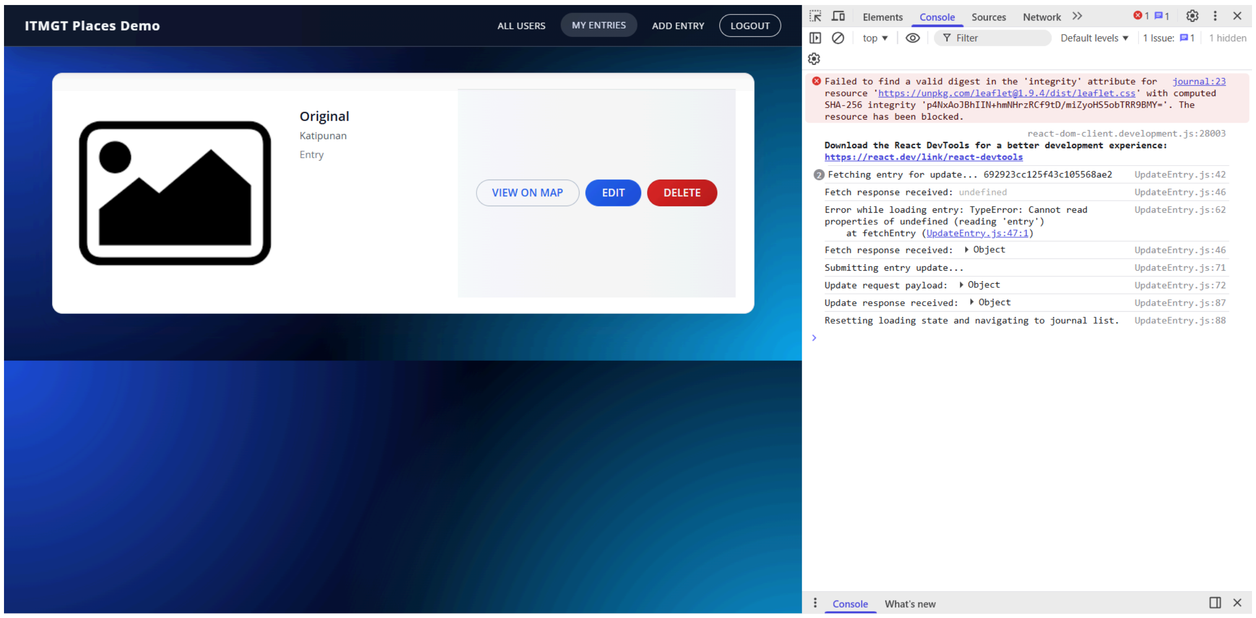
The loading UI is now controlled only by `isLoading` from `useHttpClient`, which automatically goes back to `false` after the request completes, allowing the form screen to render again.

Solution 2: Restore navigation after a successful update

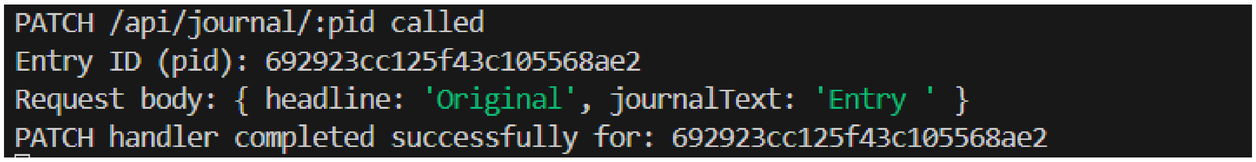
```
const responseData = await sendRequest(...);
console.log("Update response received:", responseData);
- // navigate("/" + auth.userId + "/journal");
+ navigate("/" + auth.userId + "/journal");
```

Once the PATCH succeeds, the app navigates to `"/" + auth.userId + "/journal"`, so the user sees the updated list and the spinner disappears, fully resolving Bug A.

Solution Documentation:



```
Failed to find a valid digest in the 'integrity' attribute for   journal:23
resource 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.css' with computed
SHA-256 integrity 'p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY='. 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
2 Fetching entry for update... 692923cc125f43c105568ae2   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
```

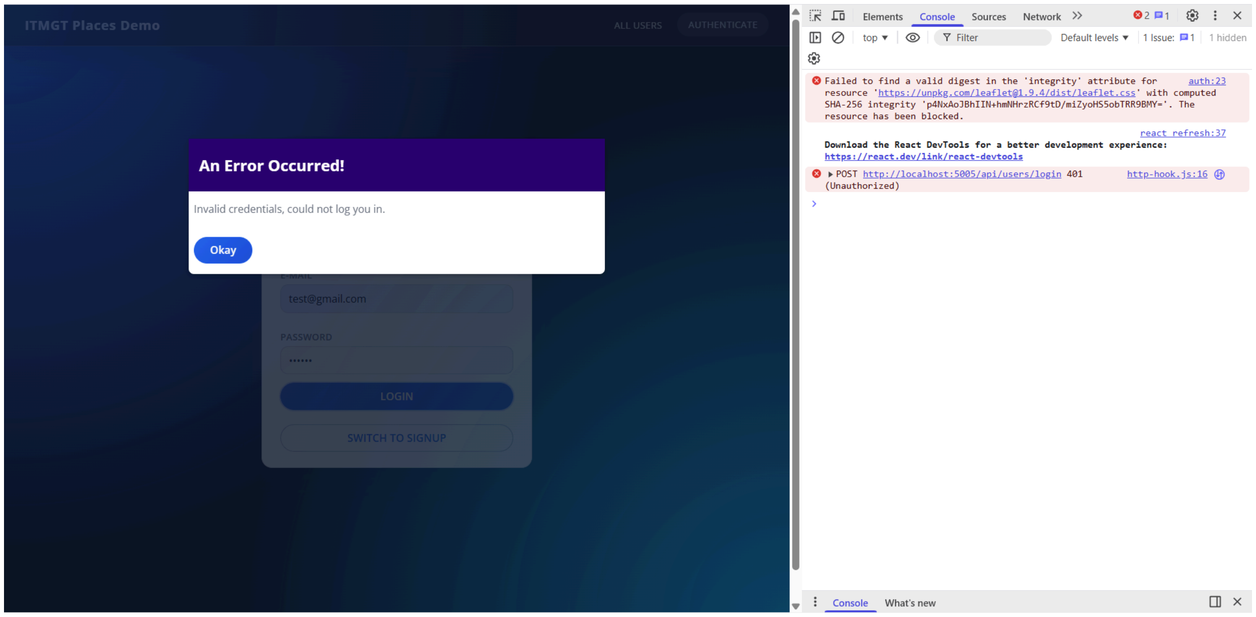


```
PATCH /api/journal/:pid called
Entry ID (pid): 692923cc125f43c105568ae2
Request body: { headline: 'Original', journalText: 'Entry ' }
PATCH handler completed successfully for: 692923cc125f43c105568ae2
```

After correcting the update flow, submitting an edit now triggers a successful `PATCH /api/journal/:pid` on the backend (as shown in the terminal logs) and the frontend logs show the response and navigation back to the journal list, confirming that the loading spinner clears and the UI no longer gets stuck.

BUG B: The Identity Crisis

Error Documentation:

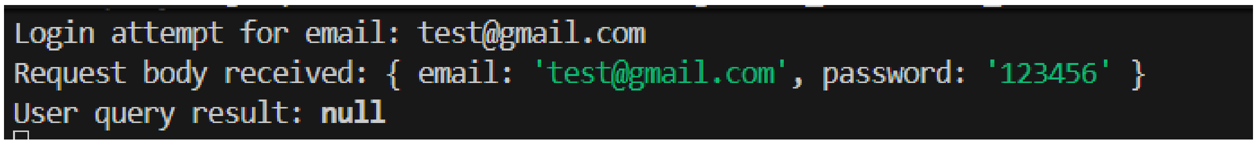


An Error Occurred!

Invalid credentials, could not log you in.

Okay

```
Failed to find a valid digest in the 'integrity' attribute for resource 'https://unpkg.com/leaflet@1.9.4/dist/leaflet.css' with computed SHA-256 integrity 'p4NxAoJBhIIN+hmNHrzRCf9tD/miZyoHS5obTRR9BMY='. The resource has been blocked.   auth:23
                                                                react_refresh:37
Download the React DevTools for a better development experience: https://react.dev/link/react-devtools
▶ POST http://localhost:5005/api/users/login 401 (Unauthorized)   http-hook.js:16
```



```
Login attempt for email: test@gmail.com
Request body received: { email: 'test@gmail.com', password: '123456' }
User query result: null
```

The frontend shows "Invalid credentials, could not log you in" after a POST to `/api/users/login` returns 401, while the backend logs show a valid email and password in the request body but `User query result: null`, confirming that the login query is looking up the user incorrectly (by `name` instead of `email`).

Console Log (Added to *login* in *users-controllers.js*)

```
const login = async (req, res, next) => {
  const { email, password } = req.body;

  console.log("Login attempt for email:", email);
  console.log("Request body received:", req.body);

  let existingUser;

  try {
    // BUG version initially used { name: email } here
    existingUser = await User.findOne({ name: email });
    console.log("User query result:", existingUser);
  } catch (err) {
    const error = new HttpError(
      "Logging in failed, please try again later.",
      500
    );
    return next(error);
  }

  if (!existingUser || existingUser.password !== password) {
    const error = new HttpError(
      "Invalid credentials, could not log you in.",
      401
    );
    return next(error);
  }

  res.json({
    message: "Logged in!",
    userId: existingUser.id,
    email: existingUser.email,
  });
};
```

These logs showed that the request body contained the correct `email` and `password`, but `User.findOne({ name: email })` returned `null`, causing every login attempt to fail with "Invalid credentials".

Error: Login query uses the wrong field (File: `users-controllers.js`, function: `login`)

```
existingUser = await User.findOne({ name: email });
console.log("User query result:", existingUser);
```

Signup stores users with an `email` field, but the login controller looked them up by `name`, so the query never matched and always returned null even for valid accounts. The terminal logs confirmed this by printing `User query result: null` for a known email.

Solution 1: Query by email instead of name

```
- existingUser = await User.findOne({ name: email });
+ existingUser = await User.findOne({ email: email });
  console.log("User query result:", existingUser);
```

The login flow now searches the `email` field, which matches both the Mongoose schema and the value sent from the frontend. For a valid account, the console now shows a full user document instead of `null`, and the login request returns 200 instead of 401.

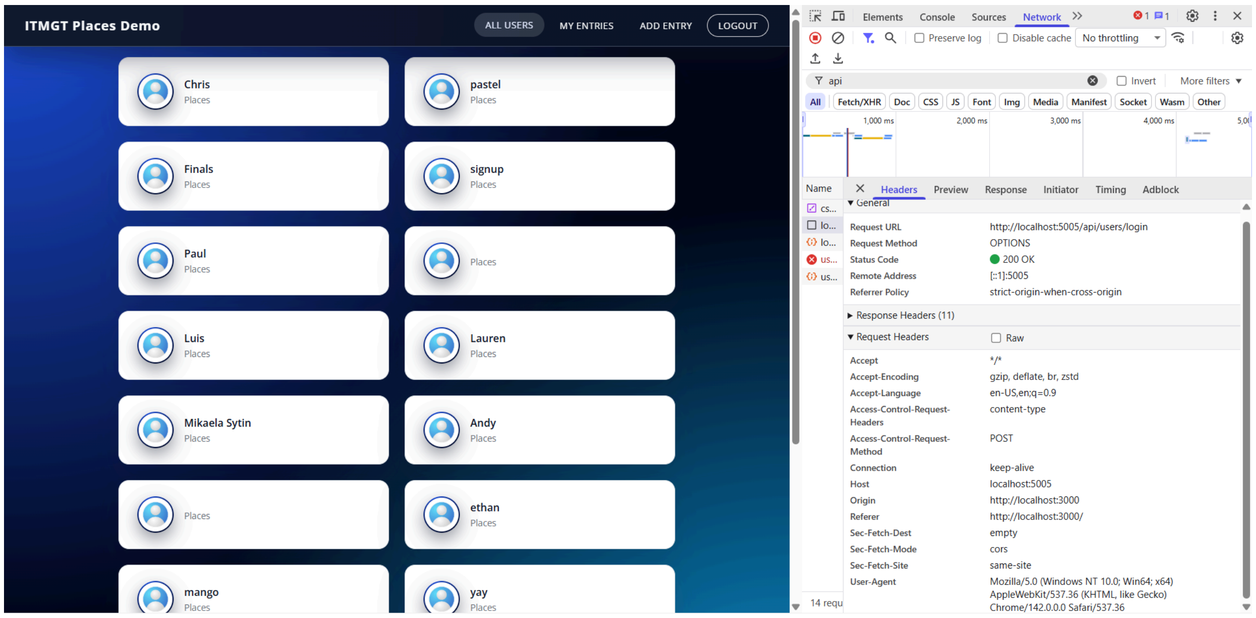
Solution 2: Keep existing credential check and success response

```
if (!existingUser || existingUser.password !== password) {
  const error = new HttpError(
    "Invalid credentials, could not log you in.",
    401
  );
  return next(error);
}
```

```
res.json({
  message: "Logged in!",
  userId: existingUser.id,
  email: existingUser.email,
});
```

With the correct query, this check now only fails for truly invalid credentials. Successful logins show "Logged in!" in the response and close the error modal in the UI, demonstrating that the identity lookup is working correctly again.

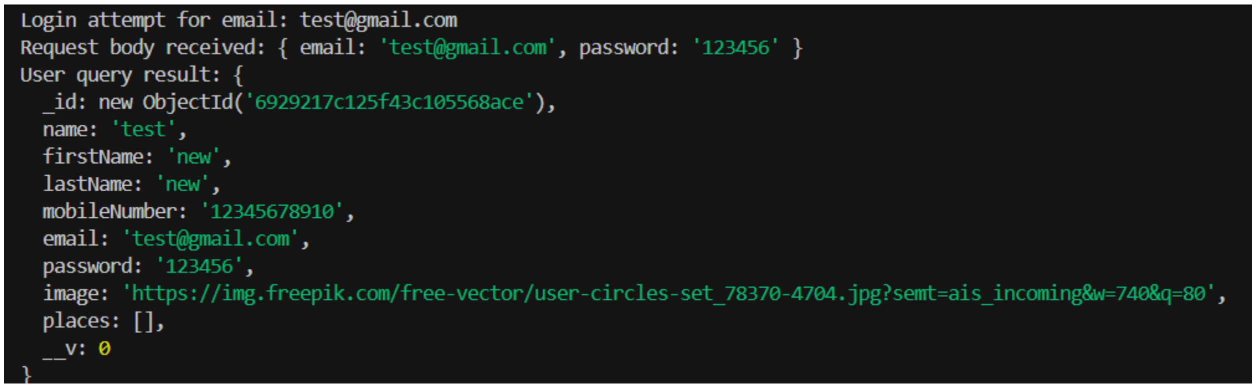
Solution Documentation



```
General
Request URL          http://localhost:5005/api/users/login
Request Method       OPTIONS
Status Code          ● 200 OK
Remote Address       [::1]:5005
Referrer Policy      strict-origin-when-cross-origin

Response Headers (11)

Request Headers                           Raw
Accept                                    */*
Accept-Encoding                           gzip, deflate, br, zstd
Accept-Language                           en-US,en;q=0.9
Access-Control-Request-Headers            content-type
Access-Control-Request-Method             POST
Connection                                keep-alive
Host                                      localhost:5005
Origin                                    http://localhost:3000
Referer                                   http://localhost:3000/
Sec-Fetch-Dest                            empty
Sec-Fetch-Mode                            cors
Sec-Fetch-Site                            same-site
User-Agent                                Mozilla/5.0 (Windows NT 10.0; Win64; x64)
                                          AppleWebKit/537.36 (KHTML, like Gecko)
                                          Chrome/142.0.0.0 Safari/537.36
```

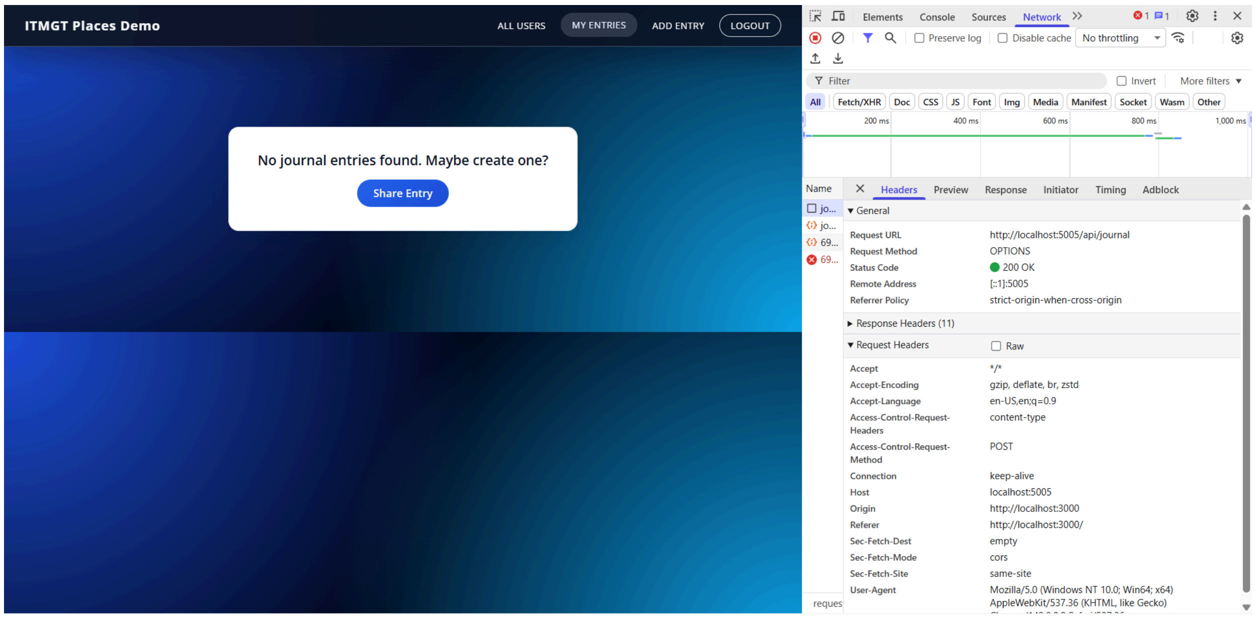


```
Login attempt for email: test@gmail.com
Request body received: { email: 'test@gmail.com', password: '123456' }
User query result: {
  _id: new ObjectId('6929217c125f43c105568ace'),
  name: 'test',
  firstName: 'new',
  lastName: 'new',
  mobileNumber: '12345678910',
  email: 'test@gmail.com',
  password: '123456',
  image: 'https://img.freepik.com/free-vector/user-circles-set_78370-4704.jpg?semt=ais_incoming&w=740&q=80',
  places: [],
  __v: 0
}
```

Fixed login flow: the Network tab now shows a successful `/api/users/login` request, and the backend logs display a populated user document for the submitted email (instead of `null`), confirming that the query in `users-controllers.js` was corrected from `{ name: email }` to `{ email: email }` and valid users can log in again.

BUG C: The Phantom List

Error Documentation:



No journal entries found. Maybe create one?

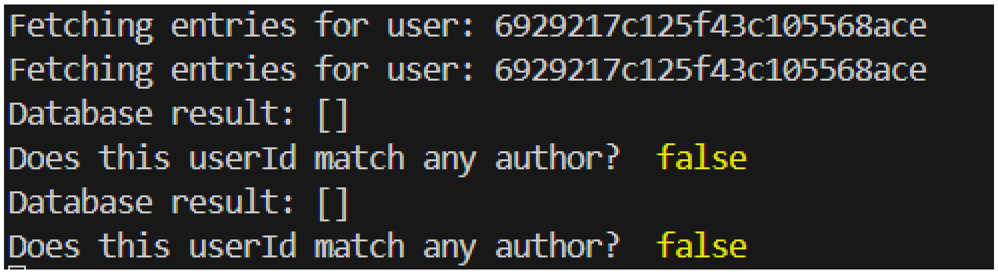
General

Request URL	http://localhost:5005/api/journal
Request Method	OPTIONS
Status Code	● 200 OK
Remote Address	[::1]:5005
Referrer Policy	strict-origin-when-cross-origin

Response Headers (11)

Request Headers

Accept	*/*
Accept-Encoding	gzip, deflate, br, zstd
Accept-Language	en-US,en;q=0.9
Access-Control-Request-Headers	content-type
Access-Control-Request-Method	POST
Connection	keep-alive
Host	localhost:5005
Origin	http://localhost:3000
Referer	http://localhost:3000/
Sec-Fetch-Dest	empty
Sec-Fetch-Mode	cors
Sec-Fetch-Site	same-site
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)



```
Fetching entries for user: 6929217c125f43c105568ace
Fetching entries for user: 6929217c125f43c105568ace
Database result: []
Does this userId match any author?  false
Database result: []
Does this userId match any author?  false
```

The "My Entries" page shows "No journal entries found" even though the `/api/journal` request returns 200, and the backend logs for `getEntriesByUserId` confirm that the query (using the wrong `creator` filter) yields an empty result array for the logged-in user

Console Log (Added to `getEntriesByUserId`)

```
const getEntriesByUserId = async (req, res, next) => {
  const userId = req.params.uid;

  console.log("Fetching entries for user:", userId);

  let entries;
  try {
    entries = await Journal.find({ creator: userId }); // BUG version
    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
  );

  // BUG: always send back an empty array
  return res.json({ entries: [] });
};
```

These logs showed that `Fetching entries for user: <userId>` printed the correct user id. `Database result: []` and `Does this userId match any author? false` appeared even after creating entries for that user. On the frontend, the "My Entries" page always showed "No journal entries found" despite successful POSTs to create entries and 200 responses from the GET endpoint

Error 1: Incorrect query filter

```
entries = await Journal.find({ creator: userId });
```

The `Journal` documents store the owning user in the `author` field (set during `createEntry`), but the query used `creator`, which does not exist in the schema. As a result, MongoDB always returned an empty array for this user id.

Error 2: Hardcoded empty response

```
return res.json({ entries: [] });
```

Even if the query ever did return data, the controller threw it away and always responded with an empty `entries` array, guaranteeing an empty list in the UI.

Solution 1: Query by the correct field (`author`)

```
- entries = await Journal.find({ creator: userId });
+ entries = await Journal.find({ author: userId });
  console.log("Database result:", entries);
```

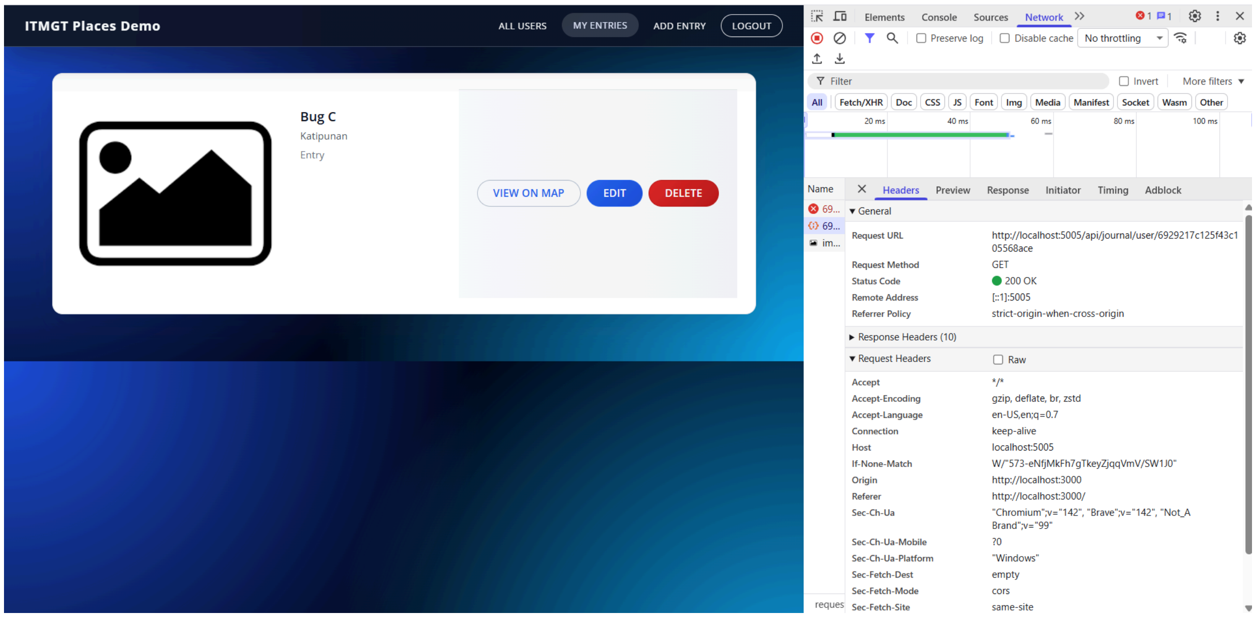
The database now searches the `author` field, which matches what `createEntry` writes. After this change, the logs showed `Database result: [ { ... author: '<userId>' ... } ]` and `Does this userId match any author? true`, confirming that entries for the current user were being retrieved.

Solution 2: Return actual database results instead of an empty array

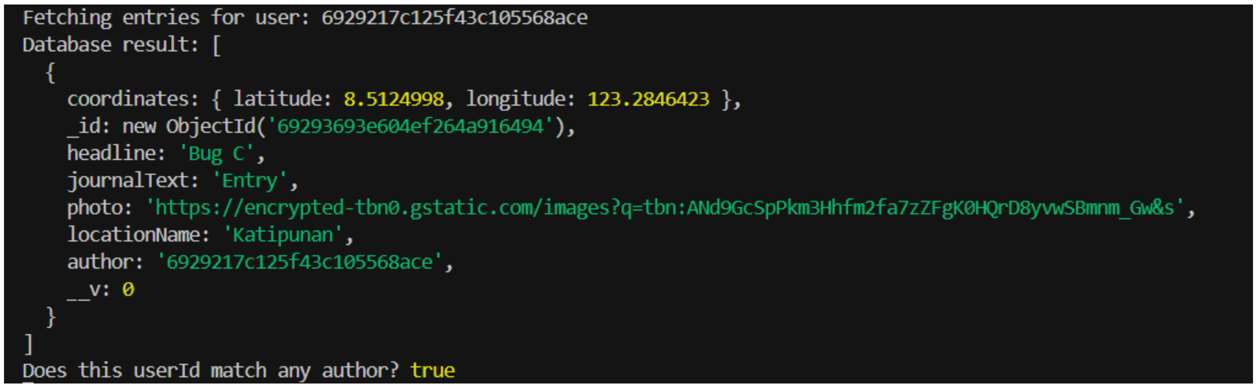
```
- return res.json({ entries: [] });
+ res.json({
+   entries: entries.map((entry) => entry.toObject({ getters: true
})),
+ });
```

The API response now includes the real list of entries from MongoDB. The "My Entries" page renders those entries (for example, the "Bug C" entry) instead of the "No journal entries found" message, demonstrating that the phantom-list behavior has been resolved.

Solution Documentation:



```
▼ General
Request URL          http://localhost:5005/api/journal/user/6929217c125f43c1
                     05568ace
Request Method       GET
Status Code          ● 200 OK
Remote Address       [::1]:5005
Referrer Policy      strict-origin-when-cross-origin
▶ Response Headers (10)
▼ Request Headers    ☐ Raw
Accept               */*
Accept-Encoding      gzip, deflate, br, zstd
Accept-Language      en-US,en;q=0.7
Connection           keep-alive
Host                 localhost:5005
If-None-Match        W/"573-eNfjMkFh7gTkeyZjqqVmV/SW1J0"
Origin               http://localhost:3000
Referer              http://localhost:3000/
Sec-Ch-Ua            "Chromium";v="142", "Brave";v="142", "Not_A
                     Brand";v="99"
Sec-Ch-Ua-Mobile     ?0
Sec-Ch-Ua-Platform   "Windows"
Sec-Fetch-Dest       empty
Sec-Fetch-Mode       cors
Sec-Fetch-Site       same-site
```



```
Fetching entries for user: 6929217c125f43c105568ace
Database result: [
  {
    coordinates: { latitude: 8.5124998, longitude: 123.2846423 },
    _id: new ObjectId('69293693e604ef264a916494'),
    headline: 'Bug C',
    journalText: 'Entry',
    photo: 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSpPkm3Hhfm2fa7zZFgK0HQrD8yvwSBmnm_Gw&s',
    locationName: 'Katipunan',
    author: '6929217c125f43c105568ace',
    __v: 0
  }
]
Does this userId match any author? true
```

After updating `getEntriesByUserId` in `journal-controllers.js` to query `Journal.find({ author: userId })` and return the actual results instead of a hardcoded empty array, the backend logs show a matching entry for the current user and the `/api/journal/user/:uid` request populates the 'My Entries' page with the saved journal instead of an empty state.