Master Team Project Fall 2024
NeuAnfang
Team 3

Member Name	Role
Zubeena Shireen Sheikh zubeena-shireen.sheikh@informatik.hs-fulda.de	Team Lead and Document Master
Muhammad Zaid Akhter	Backend Lead
Masood Ahmed Mohiuddin	Frontend Lead
Humdaan Syed	GitHub Master
Andrii Kuripka	Team Member (Frontend)

Milestone 4
06/02/2025

Date Submitted	**06/02/2025**

Content and structure for Milestone4 document for review by institutors:

1. Product Summary
2. Usability test plan
3. QA test plan
4. Code review
5. Self-check on best practices for security
6. Self-check: Adherence to original Non-functional specs

1. Product Summary

Product Name: NeuAnfang

List of committed functions:

- NeuAnfang facilitates Fulda University students to find a place they can call home in their new beginnings in Fulda.
- **Apartment Posting and Pricing:** Landlords can add detailed information about their properties.
- **Share Properties:** Ads are shareable via links. This helps landlords to increase footprint by sharing on other social media platforms.
- **Ad Status Management:** Admins can mark ads as draft, pending, active, approved, or rejected.
- **List View of Apartments:** The list view is sortable by filters and displays a summary of key information.
- **Map View of Apartments:** An interactive map shows property markers, allowing students to zoom in on specific neighbourhoods.
- **Detailed Apartment View:** The detailed view includes images and an expanded list of amenities.
- **Real-Time Chat:** Real-time chat for students and landlords to communicate with each other. It features notifications for unread messages.
- **Ad Approval by Admin:** Ads go live only after site admin reviews and approves each ad.
- **Admin Comments:** Admins can add comments to communicate with landlords to resolve issues.
- **Analytics dashboard:** Ad creators can view and analyze their Ad reception with a dashboard with various KPIs like created, active, pending ads.
- **Advanced Filtering Options:** Filters include advanced criteria and users can combine multiple filters for complex searches.
- **Student Subletting:** Students can post Sublet ads which require approval by Admin.
- **Student Account Verification:** Fulda University students can register with verification of their university email address.
- **Ad Search Feature:** Users can search for properties using natural language queries.
- **Wishlist Apartments:** Students can add apartment properties to wishlists.
- URL to page: NeuAnfang

2. Usability test plan

Usability Task Description: Search

Participants will be given the following instructions for the usability test:

- **Task 1:** Imagine you are a student looking for a place to stay in Fulda. Use the search functionality on the platform to find properties that match your preferences (e.g., price range, number of rooms).
- **Task 2:** Apply relevant filters such as budget, and amenities to refine your search results.
- **Task 3:** Look at properties of interest and review their details, noting any difficulties or missing information.

Effectiveness Measurement:

Effectiveness will be measured by the user's ability to successfully search for a property, apply filters, view property details, and contact the property owner without encountering major issues.

Efficiency Measurement:

Efficiency will be measured by the time taken to complete each task and the steps required to perform each action within the property search functionality.

Likert Scale Questions for User Satisfaction (Property Search Functionality)

Participants will use a Likert scale to assess their satisfaction with the property search functionality after completing the usability tasks. They will rate their experience based on the following criteria:

Likert Scale rating description:

1	**2**	**3**	**4**	**5**
Very Poor	Poor	Neutral	Good	Very Good

Management Student - 1

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** The property search functionality was smooth and a seamless experience.

Q2: How easy was it to search for properties and apply filters? **A2:** Searching for properties and applying filters was extremely easy.

Q3: Did you encounter any issues during the property search process? - **A3:** No issues encountered; the platform worked flawlessly.

Q4: How relevant were the search results based on your input? - **A4:** The search results were highly relevant and accurately matched my criteria.

Q5: How was your overall experience using the platform for property search? - **A5:** Overall, the platform offers a user-friendly and efficient property search experience.

Likert Scale Questions Responses:

Criteria	**Response**
Overall Experience	5/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	5/5
Inquiry Process	5/5

Management Student - 2

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Good

Q2: How easy was it to search for properties and apply filters? - **A2:** Satisfied

Q3: Did you encounter any issues during the property search process? - **A3:** No

Q4: How relevant were the search results based on your input? - **A4:** Perfect

Q5: How was your overall experience using the platform for property search? - **A5:** So good

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5
Ease of Use	5/5
Search Effectiveness	4/5
Information Clarity	5/5
Inquiry Process	3/5

News Reporter - 3

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Great Simple UI and I like it

Q2: How easy was it to search for properties and apply filters? - **A2:** Easy, Filter button and share is easily findable

Q3: Did you encounter any issues during the property search process? - **A3:** No, not an issue. Smooth process

Q4: How relevant were the search results based on your input? - **A4:** I found what I was looking for.

Q5: How was your overall experience using the platform for property search? - **A5:** Good overall.

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5

Ease of Use	5/5
Search Effectiveness	4/5
Information Clarity	5/5
Inquiry Process	4/5

German Landlord - 4

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** Good

Q2: How easy was it to search for properties and apply filters? - **A2:** Fairly easy

Q3: Did you encounter any issues during the property search process? - **A3:** No

Q4: How relevant were the search results based on your input? - **A4:** Relevant

Q5: How was your overall experience using the platform for property search? - **A5:** Good

Likert Scale Questions Responses:

Criteria	**Response**
Overall Experience	4/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	3/5
Inquiry Process	3/5

Warehouse worker - 5

Test Participant Questions:

Q1: How was your overall experience when using the property search functionality? - **A1:** It was okay

Q2: How easy was it to search for properties and apply filters? - **A2:** Easy

Q3: Did you encounter any issues during the property search process? - **A3:** Not during search, but didn't know I needed to register to initiate contact with ad owners.

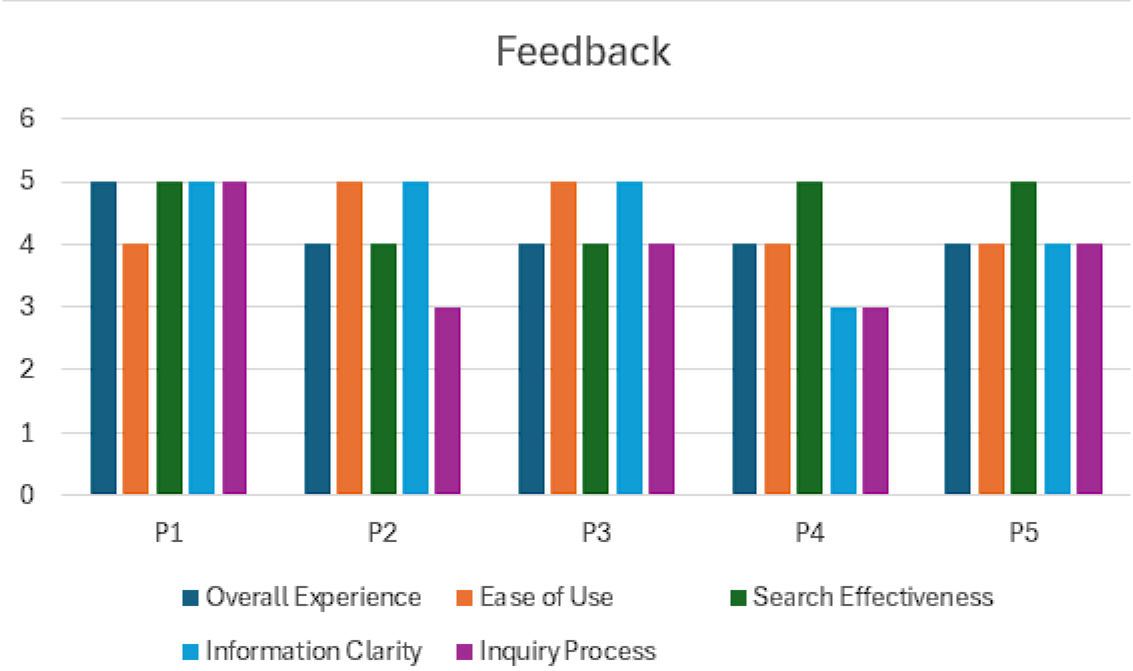
Q4: How relevant were the search results based on your input? - **A4:** Relevant, entered apartment type and matching results appeared

Q5: How was your overall experience using the platform for property search? - **A5:** Good

Likert Scale Questions Responses:

Criteria	Response
Overall Experience	4/5
Ease of Use	4/5
Search Effectiveness	5/5
Information Clarity	4/5
Inquiry Process	4/5

Feedback Chart:



3. QA test plan

Test objectives: The primary objective of this test is to ensure that the "Search" functionality on the homepage of the website operates as designed. The testing will be conducted across different hardware and browsers. Testers do not need to log in to perform this test.

HW and SW setup:

- **Hardware Setup:**
  - Windows laptops
  - Mac laptops
- **Software Setup:**
  - Latest version of Chrome browser for the Windows environment
  - Latest version of Safari browser for macOS
- **URL:** https://gdsdteam3.live/

Feature to be tested: Search functionality is tested for different sorts of user inputs which is a query text and applied filters. Initially the test is carried out for a user input text which does not match with any listings of the database. Then with a user input text which is present in the database. Next the validation of the 40 alphanumeric character limit of search text is tested. Finally, a combination of user input text and filters are verified.

Below table lists the details of tests to be carried out by the tester.

Test on Chrome

Test #	Test title	Test description	Test input	Expected output	Test results (Pass/ Fail)
1	Search with no matches	No match should result all the listings from the database	Enter a query that does not exist in the database. Eg. A string "xxxx"	Displays all the listings from the database	Pass

2	Search with matching results	Entering a query text should result in listings which include the query string in their titles	A string “2 room”	Displays listings relevant to the query	Pass
3	Search with filters	Search query by title along with filters should return results matching the filters	Enter a string “2 room”, Set the maximum price to 1600, Select filters - pets and smoking	Displays listings relevant to the query and filters	Pass
4	Price Filter	Search with both minimum and maximum price values	Enter “2200” in minimum price and “1200” in maximum price	Display error to not allow maximum price higher than minimum price	Fail
5	Character limit validation	Entering more than 40 characters should inform the user of character limit	Enter 41 “a” characters in the title field	Displays an input error	Pass

Test on Safari

Test #	Test title	Test description	Test input	Expected output	Test results (Pass/ Fail)
1	Search with no matches	No match should result all the listings from the database	Enter a query that does not exist in the database. Eg. A string “xxxx”	Displays all the listings from the database	Pass

2	Search with matching results	Entering a query text should result in listings which include the query string in their titles	A string “2 room”	Displays listings relevant to the query	Pass
3	Search with filters	Search query by title along with filters should return results matching the filters	Enter a string “2 room”, Set the maximum price to 1600, Select filters - pets and smoking	Displays listings relevant to the query and filters	Pass
4	Price Filter	Search with both minimum and maximum price values	Enter “2200” in minimum price and “1200” in maximum price	Display error to not allow maximum price higher than minimum price	Fail
5	Character limit validation	Entering more than 40 characters should inform the user of character limit	Enter 41 “a” characters in the title field	Displays an input error	Pass

4) Code Review:

Review 1

Code Writer: Muhammad Zaid Akhter
Reviewer: Humdaan Syed
Date: 05/02/2025
File reviewed: filters-section.jsx

Overview:
The filters-section.jsx file handles the property search filters for the home page. It allows users to filter properties based on title, price range, availability date, amenities), and search radius. It includes both desktop and mobile responsiveness using Mantine components. The form is managed using Mantine's useForm hook, and search parameters are updated via useSearchParams from React Router.

Strengths of the code reviewed:

1. Responsive Design: The component works well on both desktop and mobile devices. It uses Drawer for mobile filters and Paper for desktop filters, making it user-friendly on all screen sizes.

2. Good Use of Mantine Components: It uses Mantine UI elements like Stack, Title, TextInput, Select, and Checkbox, which help keep the design consistent and accessible without extra effort.

3. Dynamic URL Filtering: Filters update the URL using useSearchParams. This is helpful because users can bookmark, share filtered searches, and still see their filters when refreshing the page.

4. Separation of Concerns: The filter form's logic (Filters) is separated from the layout parts (DesktopFilters and MobileFilters). This makes the code easier to manage and update.

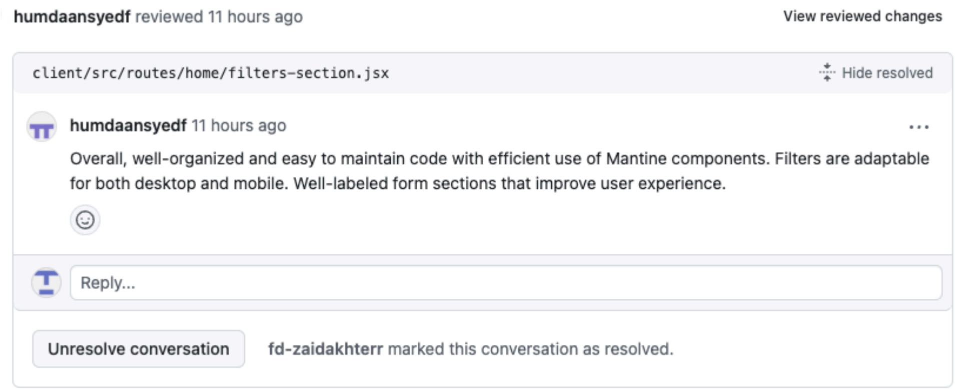
5. Reset Functionality: A simple Reset button allows users to clear all filters at once, improving the overall user experience.

Issues (if any):

Issue	Code lines in file	Screenshot
Non-Searchable MultiSelect Component	The MultiSelect component isn't searchable, which can be inconvenient when there are many amenities.	Screenshot 1

Incorrect Reset for Amenities	The pets and smoking fields are being reset to false, but MultiSelect expects an array (since amenities are handled as an array)	Screenshot 2

Github Screenshots:



humdaansyedf reviewed 11 hours ago

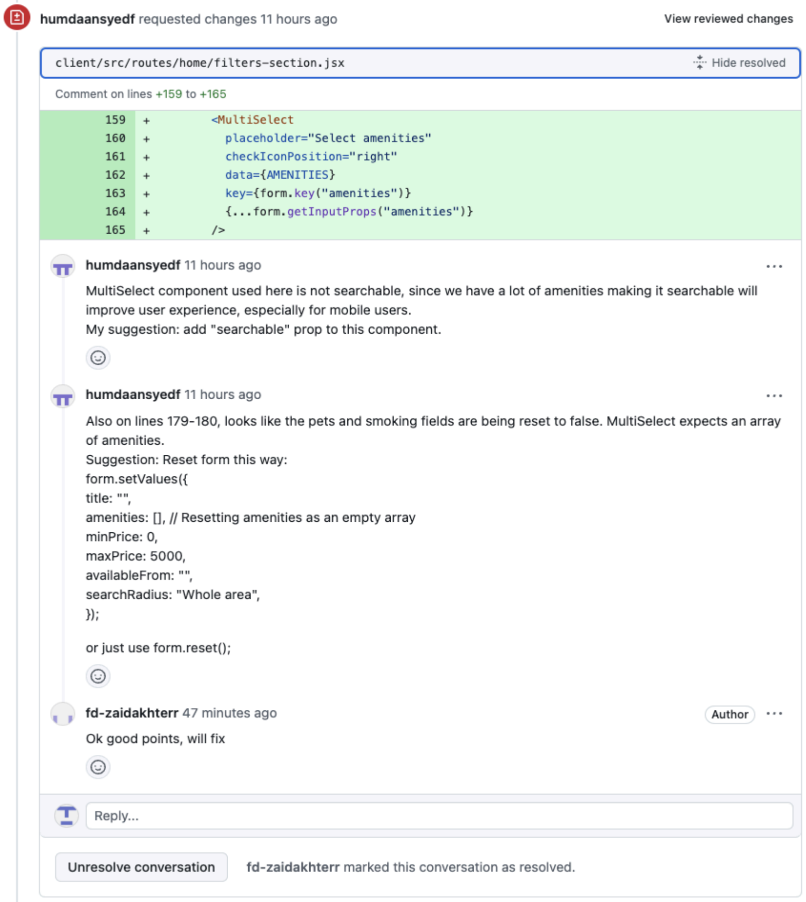
client/src/routes/home/filters-section.jsx

humdaansyedf 11 hours ago

Overall, well-organized and easy to maintain code with efficient use of Mantine components. Filters are adaptable for both desktop and mobile. Well-labeled form sections that improve user experience.

fd-zaidakhterr marked this conversation as resolved.

Screenshot 1



humdaansyedf requested changes 11 hours ago

client/src/routes/home/filters-section.jsx

Comment on lines +159 to +165

```
159 +         <MultiSelect
160 +           placeholder="Select amenities"
161 +           checkIconPosition="right"
162 +           data={AMENITIES}
163 +           key={form.key("amenities")}
164 +           {...form.getInputProps("amenities")}
165 +         />
```

humdaansyedf 11 hours ago

MultiSelect component used here is not searchable, since we have a lot of amenities making it searchable will improve user experience, especially for mobile users.
My suggestion: add "searchable" prop to this component.

humdaansyedf 11 hours ago

Also on lines 179-180, looks like the pets and smoking fields are being reset to false. MultiSelect expects an array of amenities.
Suggestion: Reset form this way:

```
form.setValues({
title: "",
amenities: [], // Resetting amenities as an empty array
minPrice: 0,
maxPrice: 5000,
availableFrom: "",
searchRadius: "Whole area",
});
```

or just use form.reset();

fd-zaidakhterr 47 minutes ago

Ok good points, will fix

fd-zaidakhterr marked this conversation as resolved.

Screenshot 2

Review 2

Code Writer: Humdaan Syed
Reviewer: Andrii Kuripka
Date: 05/02/2025
File reviewed: Wishlist and Map view

Overview:
The pull request introduces changes in multiple components, including updates to the map-view.jsx, enhancements to the wishlist.js for authorization handling, and fallback image handling for properties. The changes aim to improve user interaction on the map, address potential authorization inefficiencies, and ensure better error handling.

Strengths of the code reviewed:

1. Clear User Interaction Design: The "View Details" button in map-view.jsx allows seamless user interaction with property details. The use of Button and Paper ensures a clean and consistent UI.

2. Authorization Logic: The authorization checks in wishlist.js demonstrate awareness of secure API call handling, ensuring that unauthorized users do not flood the API with unnecessary requests.

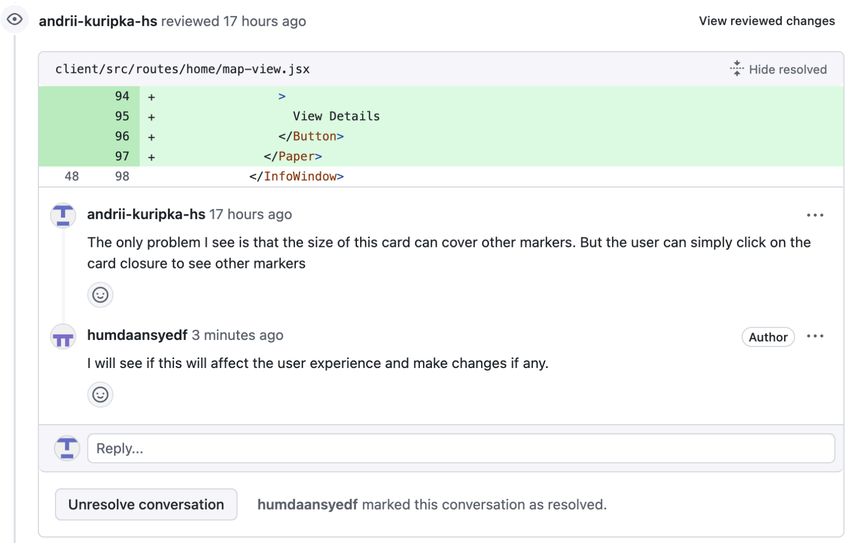
3. Fallback Image Handling: A fallback image has been added in map-view.jsx, preventing broken UI when property images are missing.

4. Responsive Improvements: The components reviewed, such as map-view.jsx, display responsiveness to varying scenarios.

Issues (if any):

Issue	Feedback	Screenshot
Marker Obstruction by InfoWindows	The addition of property cards can obstruct other markers on the map, potentially frustrating users. While users can close the card, this could negatively impact user experience if multiple markers are clustered.	Screenshot 3
Refetching on Unauthorized Requests	Unauthorized users trigger excessive API refetches, which is inefficient and may lead to server strain.	Screenshot 4

Github Screenshots:



andrii-kuripka-hs reviewed 17 hours ago

client/src/routes/home/map-view.jsx

```
+                >
+                  View Details
+                </Button>
+              </Paper>
           </InfoWindow>
```

andrii-kuripka-hs 17 hours ago

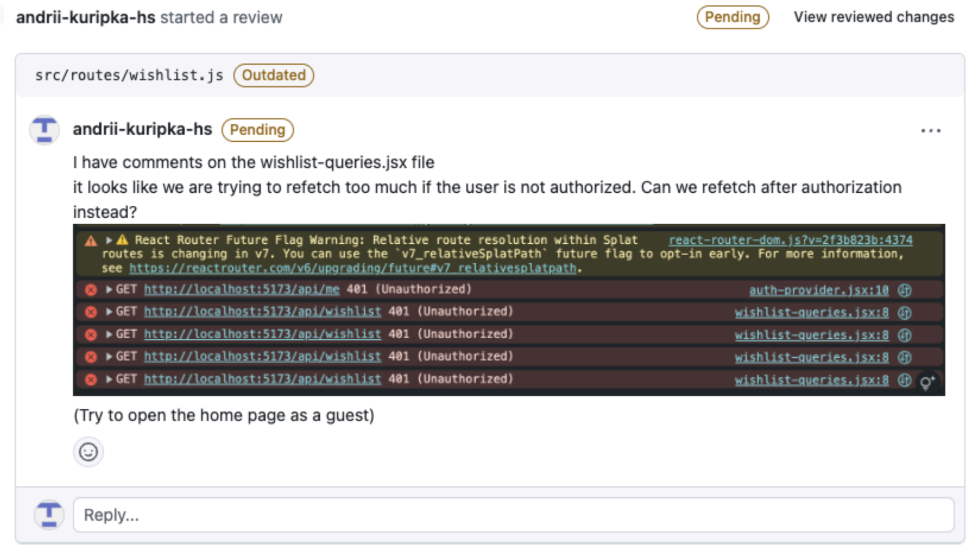
The only problem I see is that the size of this card can cover other markers. But the user can simply click on the card closure to see other markers

humdaansyedf 3 minutes ago

I will see if this will affect the user experience and make changes if any.

humdaansyedf marked this conversation as resolved.

Screenshot 3



andrii-kuripka-hs started a review

src/routes/wishlist.js

andrii-kuripka-hs

I have comments on the wishlist-queries.jsx file
it looks like we are trying to refetch too much if the user is not authorized. Can we refetch after authorization instead?

```
⚠ ▶⚠ React Router Future Flag Warning: Relative route resolution within Splat     react-router-dom.js?v=2f3b823b:4374
  routes is changing in v7. You can use the `v7_relativeSplatPath` future flag to opt-in early. For more information,
  see https://reactrouter.com/v6/upgrading/future#v7_relativesplatpath.
⊗ ▶GET http://localhost:5173/api/me 401 (Unauthorized)                              auth-provider.jsx:10
⊗ ▶GET http://localhost:5173/api/wishlist 401 (Unauthorized)                    wishlist-queries.jsx:8
⊗ ▶GET http://localhost:5173/api/wishlist 401 (Unauthorized)                    wishlist-queries.jsx:8
⊗ ▶GET http://localhost:5173/api/wishlist 401 (Unauthorized)                    wishlist-queries.jsx:8
⊗ ▶GET http://localhost:5173/api/wishlist 401 (Unauthorized)                    wishlist-queries.jsx:8
```

(Try to open the home page as a guest)

Screenshot 4

Review 3

Code Writer: Andrii Kuripka
Reviewer: Zubeena Shireen Sheikh
Date: 30/01/2025
File reviewed: Edit Property Page

Overview:
This pull request introduces multiple updates across the project, including:

- Preloading existing images in the ImageUploader component.
- Fixing image updates in the backend creator.js.
- Resolving a 500 error in edit-ad-page.jsx due to incorrect date formatting.

These changes aim to enhance the user experience by fixing bugs and improving functionality, particularly in property management and editing.

Strengths of the code reviewed:

1. Preloading Existing Images: In the ImageUploader component, the ability to load and display existing images from the database provides a smoother user experience for updating property listings.

2. Bug Fix for Date Format: The fix for the date formatting issue in edit-ad-page.jsx correctly uses ISO string formatting to prevent server-side 500 errors. The use of value.toISOString().substring(0, 10) ensures that only valid date strings are sent to the backend.

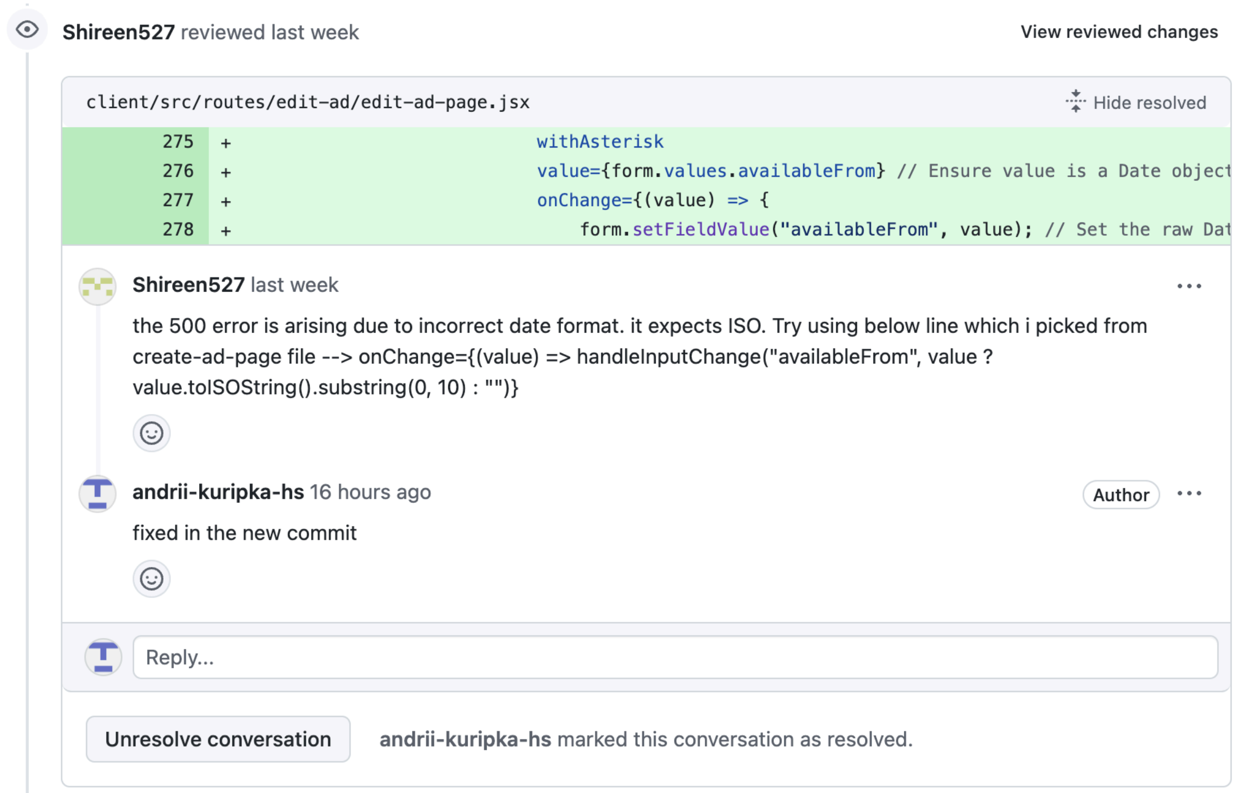
3. Effort to Update Images: Address image updates in the creator.js backend route. This shows a focus on ensuring backend and frontend synchronization for image uploads.

Issues (if any):

Issue	Feedback	Screenshot
Date Formatting in edit-ad-page.jsx	The fix addresses the immediate 500 error. However, the logic assumes value is always a valid Date object. Edge cases, such as null or invalid dates, might still cause issues.	Screenshot 5
Missing Logic for Image Updates	While the property update works, the lack of image update logic means users cannot modify the images associated	Feedback during 1-1 session

	with a property. This can lead to incomplete or outdated property listings.	

Github Screenshots:



Shireen527 reviewed last week

client/src/routes/edit-ad/edit-ad-page.jsx

```
+                               withAsterisk
+                               value={form.values.availableFrom} // Ensure value is a Date object
+                               onChange={(value) => {
+                                   form.setFieldValue("availableFrom", value); // Set the raw Dat
```

Shireen527 last week

the 500 error is arising due to incorrect date format. it expects ISO. Try using below line which i picked from create-ad-page file --> onChange={(value) => handleInputChange("availableFrom", value ? value.toISOString().substring(0, 10) : "")}

andrii-kuripka-hs 16 hours ago

fixed in the new commit

andrii-kuripka-hs marked this conversation as resolved.

Screenshot 5

Review 4

Code Writer: Zubeena Shireen Sheikh
Reviewer: Masood Ahmed Mohiuddin
Date: 05/02/2025
PR reviewed: Chat: refactoring, notifications

Overview:
The work on chat refactoring, and notifications handles socket events related to chat functionality, including joining rooms, sending messages, fetching chat history, and retrieving user interactions. It integrates Prisma for database operations and ensures real-time updates using Socket.IO. Additionally, add notification logic for chat interactions.

Strengths of the code reviewed:

1. Responsive Design: The component works well on both desktop and mobile devices. It uses Drawer for mobile filters and Paper for desktop filters, making it user-friendly on all screen sizes.

2. Good Use of Mantine Components: It uses Mantine UI elements like Stack, Title, TextInput, Select, and Checkbox, which help keep the design consistent and accessible without extra effort.

3. Database Interaction with Prisma: Queries are structured well with Prisma for fetching, creating, and updating chat and message records.

4. Good Real-time Messaging Implementation: The chat system efficiently uses socket.join(chatId) and socket.broadcast.to(chatId).emit() to handle user interactions in real-time. Notifications are triggered as expected.

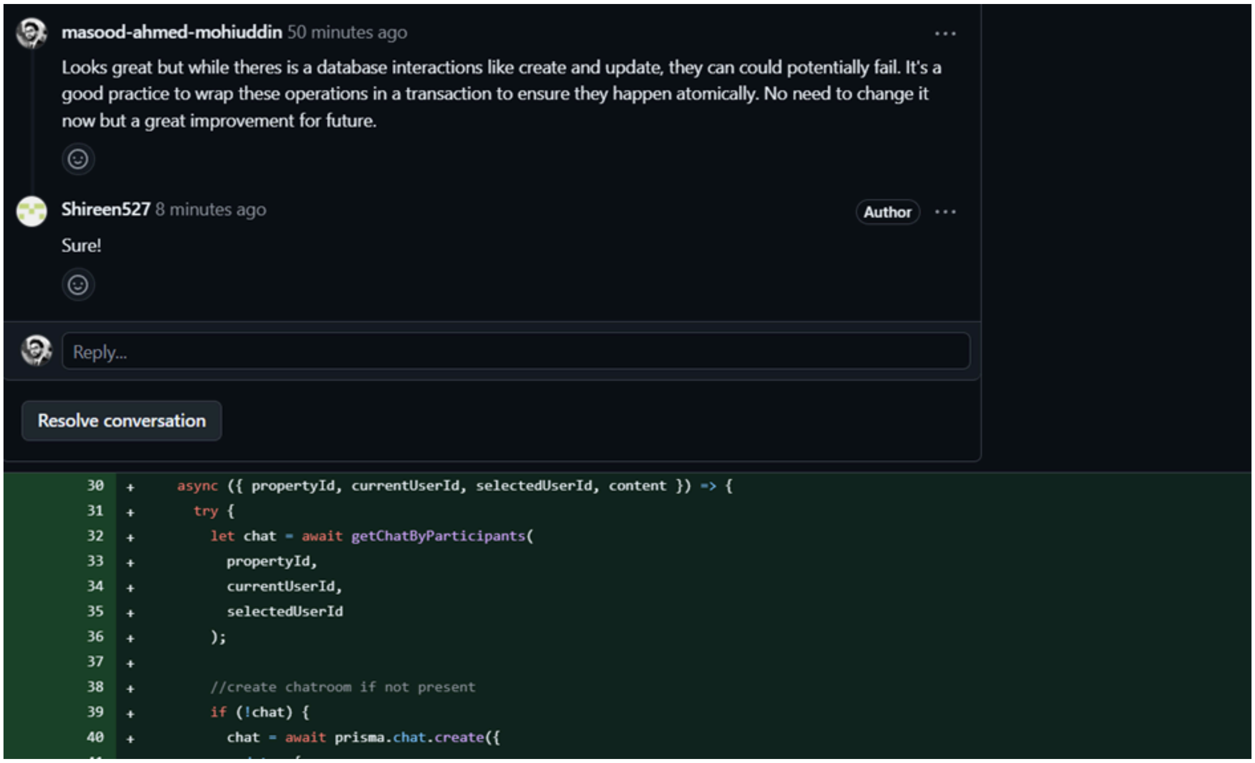
5. Good Use of React Query for Fetching Users: useQuery is implemented well to fetch chat users efficiently, improving API call structure.

6. Code Readability & Maintainability: Proper structuring and indentation.
   Clear separation of concerns for handling different socket events.

Issues (if any):

Issue	Feedback	Screenshot
Database Transactions. **Create** and **update** operations can fail, leading to partial state inconsistency.	Wrap database operations inside Prisma transactions	Screenshot 6
Performance Concern in Message Rendering. The Stack component renders all messages, even if not visible, causing potential UI lag.	Implement lazy loading or using libraries for better performance.	Screenshot 7
Fetching All Messages at Once. The query fetches all messages in a chat, which may slow down large conversations.	Implement pagination with skip and take parameters for fetching messages in chunks.	Screenshot 8
Queries like useChatUsers fetch all user data without caching.	Add staleTime or cacheTime options in useQuery to prevent unnecessary refetching.	Screenshot 9

Github Screenshots:



masood-ahmed-mohiuddin 50 minutes ago

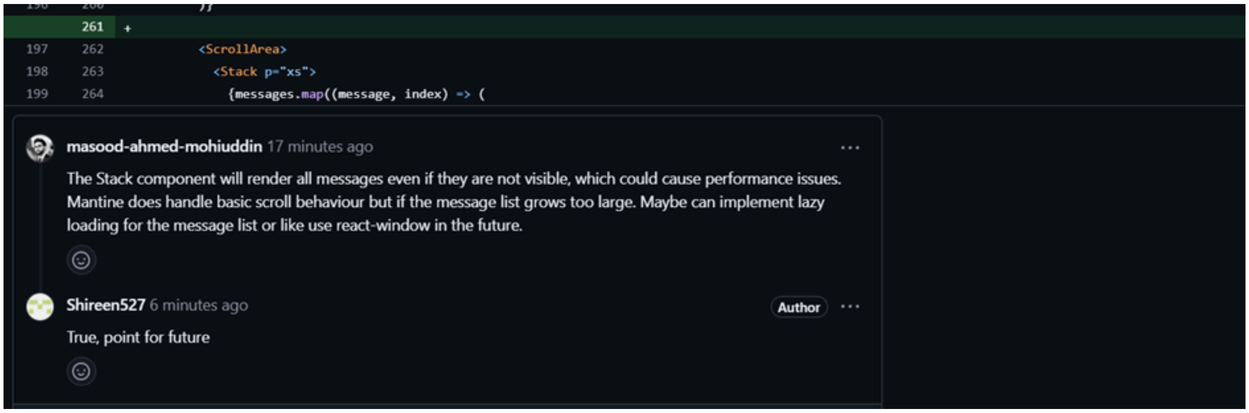
Looks great but while theres is a database interactions like create and update, they can could potentially fail. It's a good practice to wrap these operations in a transaction to ensure they happen atomically. No need to change it now but a great improvement for future.

Shireen527 8 minutes ago Author

Sure!

```
+     async ({ propertyId, currentUserId, selectedUserId, content }) => {
+       try {
+         let chat = await getChatByParticipants(
+           propertyId,
+           currentUserId,
+           selectedUserId
+         );
+
+         //create chatroom if not present
+         if (!chat) {
+           chat = await prisma.chat.create({
```

Screenshot 6



```
          })
+
          <ScrollArea>
            <Stack p="xs">
              {messages.map((message, index) => (
```

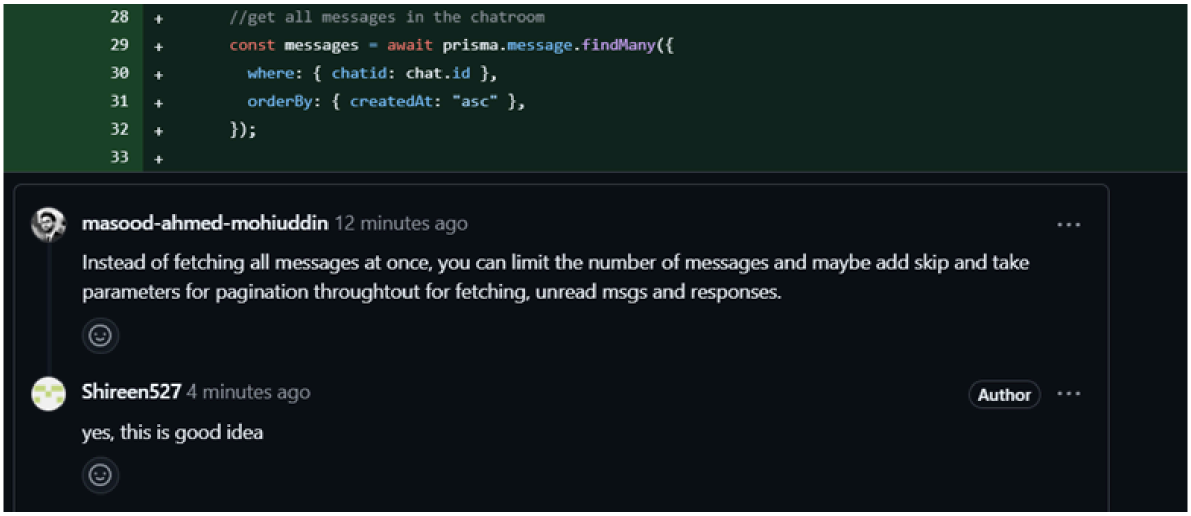
masood-ahmed-mohiuddin 17 minutes ago

The Stack component will render all messages even if they are not visible, which could cause performance issues. Mantine does handle basic scroll behaviour but if the message list grows too large. Maybe can implement lazy loading for the message list or like use react-window in the future.

Shireen527 6 minutes ago Author

True, point for future

Screenshot 7



```
+     //get all messages in the chatroom
+     const messages = await prisma.message.findMany({
+       where: { chatid: chat.id },
+       orderBy: { createdAt: "asc" },
+     });
+
```

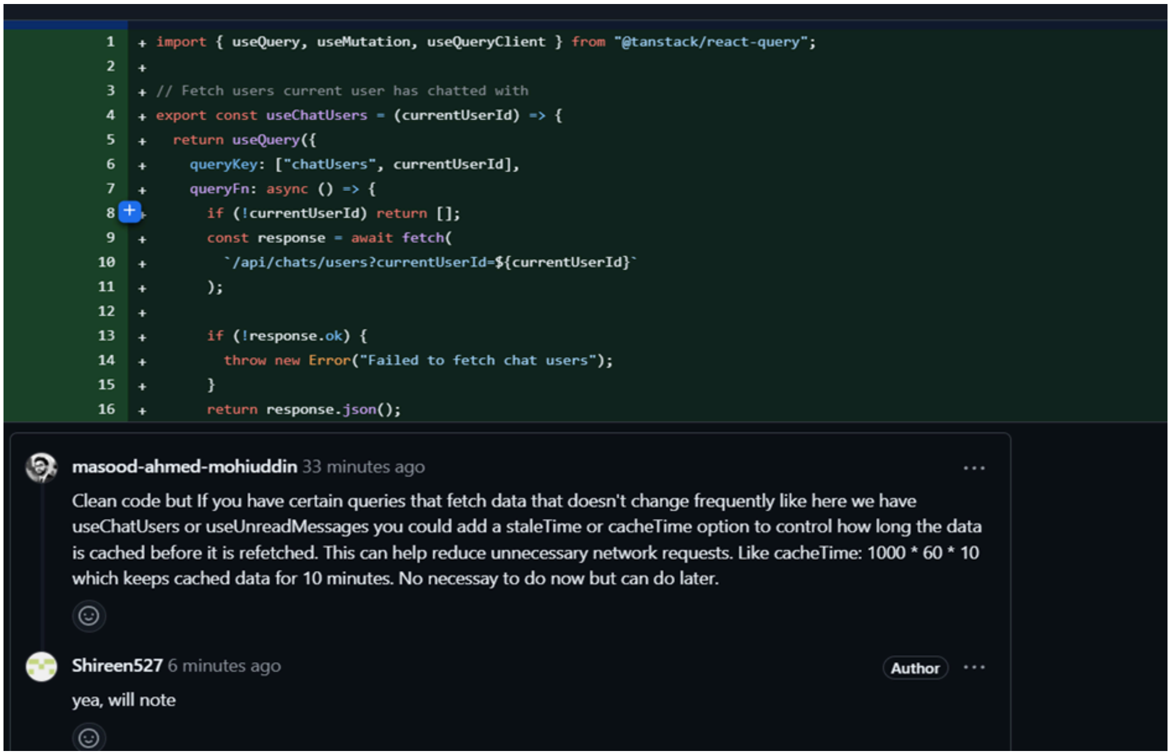
masood-ahmed-mohiuddin 12 minutes ago

Instead of fetching all messages at once, you can limit the number of messages and maybe add skip and take parameters for pagination throughtout for fetching, unread msgs and responses.

Shireen527 4 minutes ago Author

yes, this is good idea

Screenshot 8



```
+ import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
+
+ // Fetch users current user has chatted with
+ export const useChatUsers = (currentUserId) => {
+   return useQuery({
+     queryKey: ["chatUsers", currentUserId],
+     queryFn: async () => {
+       if (!currentUserId) return [];
+       const response = await fetch(
+         `/api/chats/users?currentUserId=${currentUserId}`
+       );
+
+       if (!response.ok) {
+         throw new Error("Failed to fetch chat users");
+       }
+       return response.json();
```

masood-ahmed-mohiuddin 33 minutes ago

Clean code but If you have certain queries that fetch data that doesn't change frequently like here we have useChatUsers or useUnreadMessages you could add a staleTime or cacheTime option to control how long the data is cached before it is refetched. This can help reduce unnecessary network requests. Like cacheTime: 1000 * 60 * 10 which keeps cached data for 10 minutes. No necessay to do now but can do later.

Shireen527 6 minutes ago Author

yea, will note

Screenshot 9

Review 5

Code Writer: Masood Ahmed Mohiuddin
Reviewer: Muhammad Zaid Akhter
Date: 28/01/2025
File reviewed: Student document upload + UI changes

Overview:
This PR introduces several improvements and optimizations for handling document-related functionality, including:

- Fixes to prevent generating signed URLs for all documents on fetch.
- Implementation of a new "View" button in the UI that requests signed URLs dynamically for individual documents.
- Adjustments to the CORS setup to differentiate between development and production environments.
- Refactoring of backend logic to improve efficiency and reduce unnecessary S3 requests.

While the functionality is solid, there is room for further refinement, especially in aligning with existing project patterns (tanstack-query) and improving code reusability. Once these changes are addressed, the PR will significantly improve performance and maintainability.

Strengths of the code reviewed:

1. Signed URL Optimization: Eliminating the generation of signed URLs during the fetch of all documents reduces unnecessary overhead and S3 usage.

2. Separation of Concerns: The updated logic adheres to best practices by separating document metadata fetching and URL generation. This ensures the backend is more efficient and avoids overloading the frontend with redundant data.

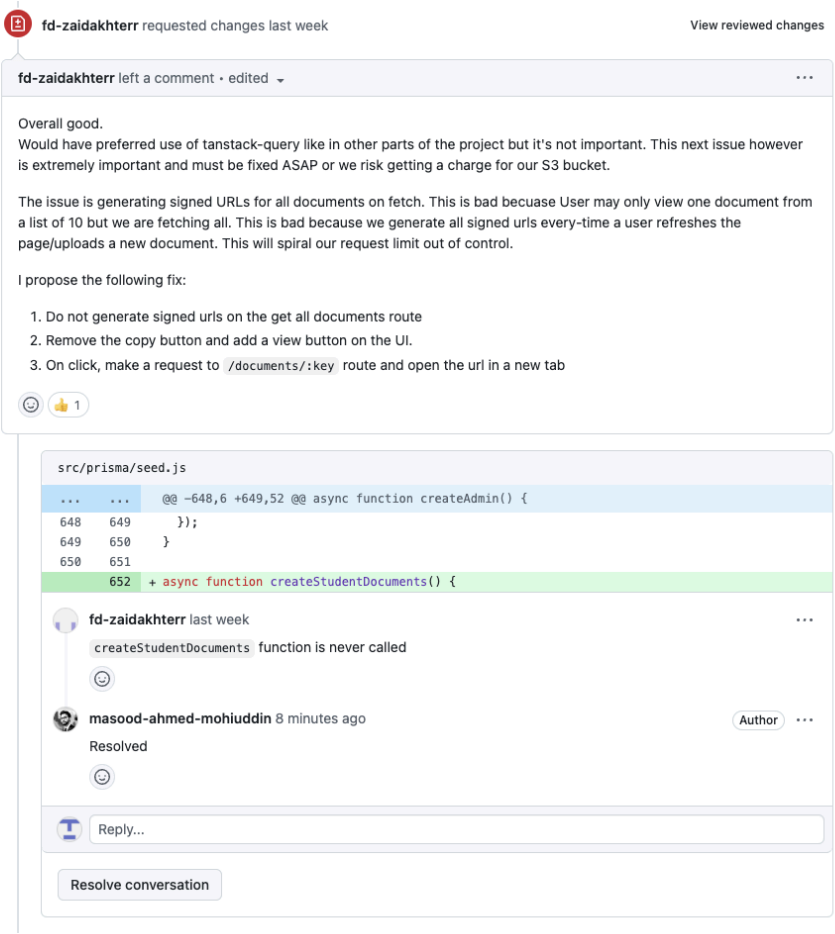
3. Improved CORS Setup: Adding an environment-based configuration for CORS prevents potential issues in production environments, ensuring secure and controlled API access.

4. Dynamic UI Enhancements: The "View" button now fetches signed URLs dynamically, addressing the limitation of short-lived signed URLs and enhancing user experience.

Issues (if any):

Issue	Feedback	Screenshot
Never called Function	The createStudentDocuments function was added but never called, making it redundant in the current implementation.	Screenshot 10
“Copy” Button Replacement	A better approach was suggested: replace the “Copy” button with a “View” button that makes a call to the /documents/:key route to fetch a signed URL dynamically and open it in a new tab.	Screenshot 11
Backend Signed URL Optimization	It was recommended to send document metadata only (without signed URLs) and generate the signed URL dynamically when requested via the /documents/:key route.	Screenshot 12

Github Screenshots:



fd-zaidakhterr requested changes last week

fd-zaidakhterr left a comment • edited

Overall good.
Would have preferred use of tanstack-query like in other parts of the project but it's not important. This next issue however is extremely important and must be fixed ASAP or we risk getting a charge for our S3 bucket.

The issue is generating signed URLs for all documents on fetch. This is bad becuase User may only view one document from a list of 10 but we are fetching all. This is bad because we generate all signed urls every-time a user refreshes the page/uploads a new document. This will spiral our request limit out of control.

I propose the following fix:

1. Do not generate signed urls on the get all documents route
2. Remove the copy button and add a view button on the UI.
3. On click, make a request to `/documents/:key` route and open the url in a new tab

👍 1

src/prisma/seed.js

```
...   ...   @@ -648,6 +649,52 @@ async function createAdmin() {
648   649       });
649   650     }
650   651
      652 + async function createStudentDocuments() {
```

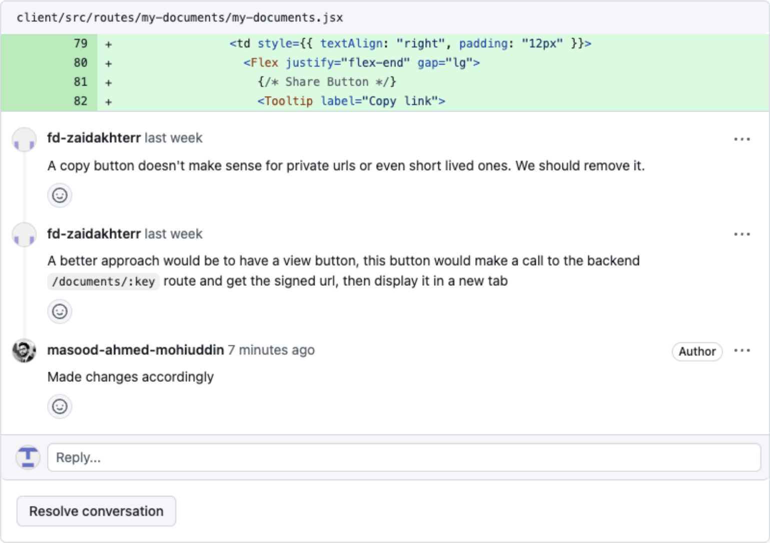
fd-zaidakhterr last week

`createStudentDocuments` function is never called

masood-ahmed-mohiuddin 8 minutes ago — Author

Resolved

Screenshot 10



client/src/routes/my-documents/my-documents.jsx

```
79  +              <td style={{ textAlign: "right", padding: "12px" }}>
80  +                <Flex justify="flex-end" gap="lg">
81  +                  {/* Share Button */}
82  +                  <Tooltip label="Copy link">
```

fd-zaidakhterr last week

A copy button doesn't make sense for private urls or even short lived ones. We should remove it.

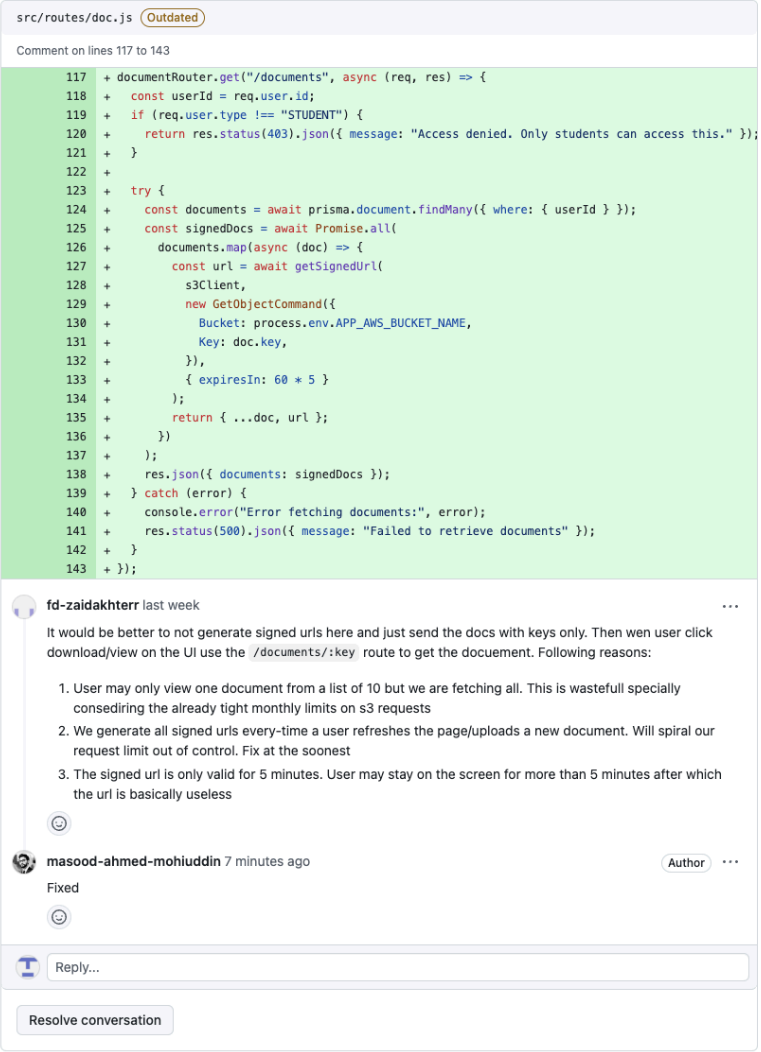
fd-zaidakhterr last week

A better approach would be to have a view button, this button would make a call to the backend `/documents/:key` route and get the signed url, then display it in a new tab

masood-ahmed-mohiuddin 7 minutes ago

Made changes accordingly

Screenshot 11



src/routes/doc.js

Comment on lines 117 to 143

```
117 + documentRouter.get("/documents", async (req, res) => {
118 +   const userId = req.user.id;
119 +   if (req.user.type !== "STUDENT") {
120 +     return res.status(403).json({ message: "Access denied. Only students can access this." });
121 +   }
122 +
123 +   try {
124 +     const documents = await prisma.document.findMany({ where: { userId } });
125 +     const signedDocs = await Promise.all(
126 +       documents.map(async (doc) => {
127 +         const url = await getSignedUrl(
128 +           s3Client,
129 +           new GetObjectCommand({
130 +             Bucket: process.env.APP_AWS_BUCKET_NAME,
131 +             Key: doc.key,
132 +           }),
133 +           { expiresIn: 60 * 5 }
134 +         );
135 +         return { ...doc, url };
136 +       })
137 +     );
138 +     res.json({ documents: signedDocs });
139 +   } catch (error) {
140 +     console.error("Error fetching documents:", error);
141 +     res.status(500).json({ message: "Failed to retrieve documents" });
142 +   }
143 + });
```

fd-zaidakhterr last week

It would be better to not generate signed urls here and just send the docs with keys only. Then wen user click download/view on the UI use the `/documents/:key` route to get the docuement. Following reasons:

1. User may only view one document from a list of 10 but we are fetching all. This is wastefull specially consediring the already tight monthly limits on s3 requests
2. We generate all signed urls every-time a user refreshes the page/uploads a new document. Will spiral our request limit out of control. Fix at the soonest
3. The signed url is only valid for 5 minutes. User may stay on the screen for more than 5 minutes after which the url is basically useless

masood-ahmed-mohiuddin 7 minutes ago

Fixed

Screenshot 12

5) Self-check on best practices for security

Major Assets and Their Protection:

1. Database (AWS RDS):
   - Threats:
     - Unauthorized access by external attackers
     - SQL injection attacks
   - Protection:
     - The secure database is hosted on RDS with public access blocked.
     - Access is strictly limited to the designated EC2 instance through a dedicated VPC.
     - Passwords are hashed using @node-rs/argon2
2. Object Storage (AWS S3):
   - Threats:
     - Exposing sensitive private student data due to misconfigured policies
     - Unauthorized data retrieval
   - Protection:
     - S3 is configured with a policy that allows public read-only access strictly for designated public content while keeping private user files inaccessible via public endpoints.
     - Access is managed and audited through AWS IAM roles and bucket policies.
3. Application Server and API Layer:
   - Threats:
     - Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF)
     - Injection attacks and unauthorized data exposure
   - Protection:
     - Robust input validation on the frontend using mantine forms
     - Strict CORs policies
     - Robust validation on the backend using zod schemas
     - Escaping user input via prisma
     - SSL Certificates configured on the server using Let's Encrypt

Password Encryption:

Passwords stored in the DB are encrypted using secure hashing algorithms (@node-rs/argon2) ensuring that even if the database is compromised, plaintext passwords are not retrievable.

Input Validation:

All inputs, including the search bar input (expected to be up to 40 alphanumeric characters), are validated rigorously. See example code for registration form validation using mantine forms below:

```
validate: {
  name: (value) => {
    return value.length < 2 ? "Name must have at least 3 letters" : null;
  },
  email: (value, values) => {
    const isEmail = /^\S+@\S+$/.test(value);
    if (!isEmail) {
      return "Invalid email";
    }
    if (values.type === "STUDENT" && !value.endsWith("hs-fulda.de")) {
      return "Please use your university email";
    }
    return null;
  },
  phone: (value) => {
    if (!value) {
      return null;
    }
    const isGermanMobile = /^(\+49|0049|0)(15|16|17)\d{8,9}$/.test(value);
    if (!isGermanMobile) {
      return "Invalid phone number";
    }
    return null;
  },
  password: (value) => {
    return value.length < 8 ? "Password must have at least 8 letters" : null;
  },
  confirmPassword: (value, values) => {
    return value !== values.password ? "Passwords do not match" : null;
  },
}
```

6. Self-check: Adherence to original Non-functional specs

1. Application shall be developed, tested, and deployed using tools and servers approved by Class CTO and as agreed in Milestone 0. Application delivery shall be from the chosen cloud
   Server **DONE**
2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers **DONE**
3. All or selected application functions must render well on mobile devices **DONE**
4. Data shall be stored in the database on the team's deployment cloud server. **DONE**
5. Full resolution free media shall be downloadable directly, and full resolution media for selling shall be obtained after contacting the seller/owner. **DONE**
6. No more than 50 concurrent users shall be accessing the application at any time. **DONE**
7. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. **DONE**
8. The language used shall be English (no localization needed) **DONE**
9. Application shall be very easy to use and intuitive **DONE**
10. Application should follow established architecture patterns **DONE**
11. Application code and its repository shall be easy to inspect and maintain **DONE**
12. Google Analytics shall be used (optional for Fulda teams) **NA**
13. No email clients shall be allowed. **DONE**
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. **DONE**
15. Site security: basic best practices shall be applied (as covered in the class) for main data items **DONE**
16. Application shall be media rich (images, video etc.). Media formats shall be standard as used in the market today **DONE**
17. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development **DONE**
18. For code development and management, as well as documentation like formal milestones required in the class, each team shall use their own GitHub to be set up by class instructors and started by each team during Milestone 0 **DONE**
19. The application UI (WWW and mobile) shall prominently display the following exact text on all pages "Fulda University of Applied Sciences Software Engineering Project, Fall 2024 For Demonstration Only" at the top of the WWW page. (Important to not confuse this with a real application). **DONE**