Collaborative Note-Taking Web Application Report

Table of Contents

Introduction

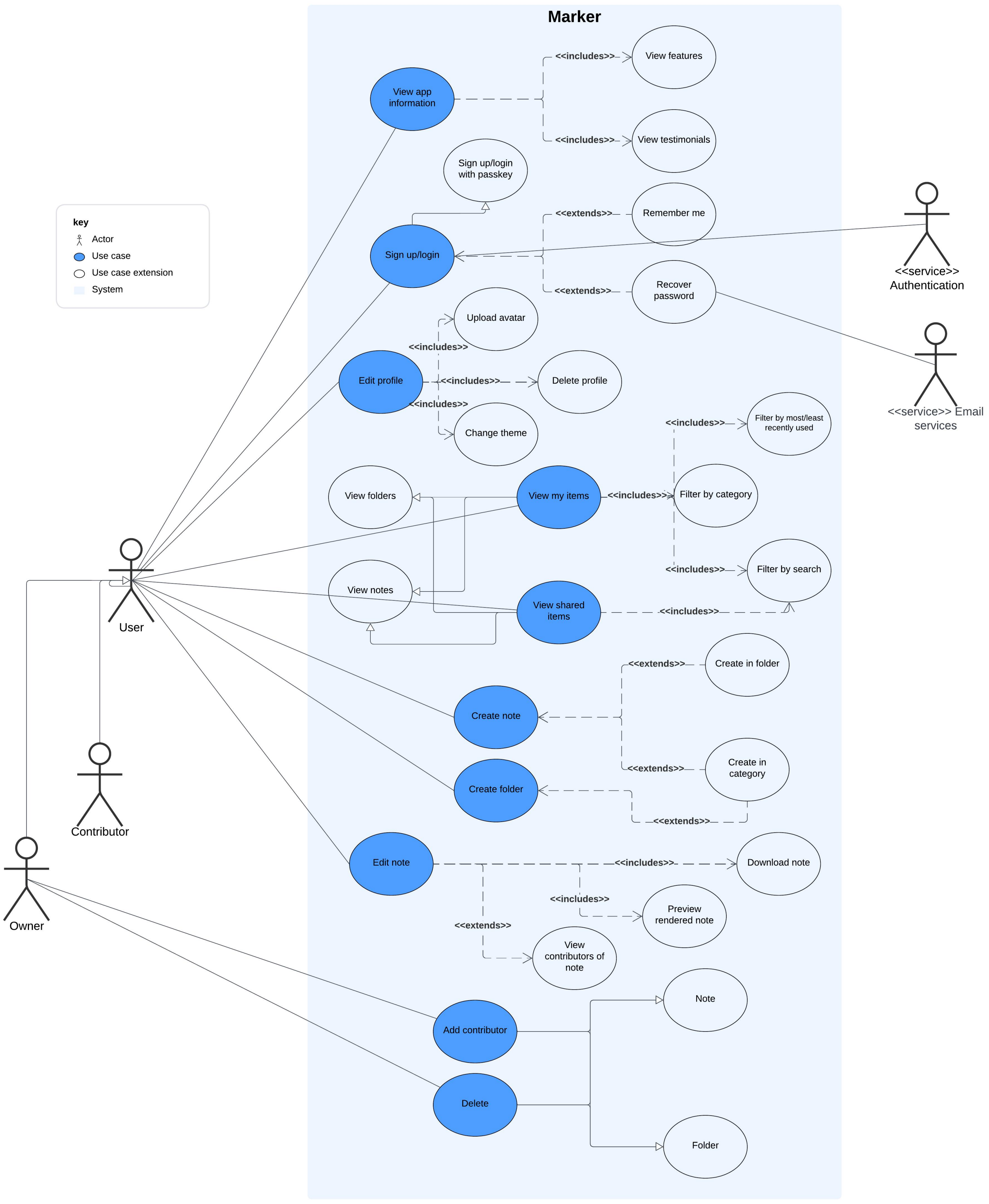
Our collaborative note-taking web application, Marker, is designed to provide users with a seamless and efficient platform for creating, editing, and sharing notes in real-time. Built with modern web technologies, our application offers a responsive and accessible user interface, robust backend services, and secure data management.

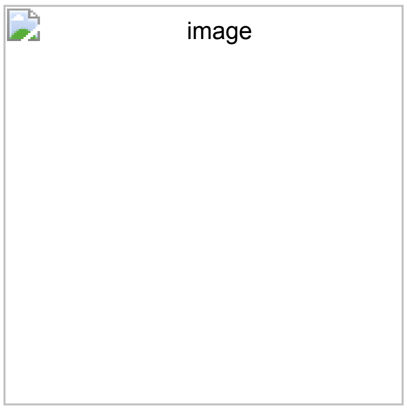
Key features of our application include:

- User authentication and account management
- Markdown-supported note creation and editing
- Real-time collaboration on shared notes
- Categorization and search functionality for notes
- Responsive design for various devices and screen sizes

This report outlines the technical architecture, design decisions, and implementation details of our application.

Use case diagram


Marker
key
Actor
Use case
Use case extension
System
View app information
<<includes>>
View features
<<includes>>
View testimonials
Sign up/login with passkey
Sign up/login
<<extends>>
Remember me
<<extends>>
Recover password
<<service>> Authentication
<<service>> Email services
Upload avatar
<<includes>>
Edit profile
<<includes>>
Delete profile
<<includes>>
Change theme
<<includes>>
Filter by most/least recently used
View folders
View my items
<<includes>>
Filter by category
View notes
<<includes>>
Filter by search
View shared items
<<includes>>
User
<<extends>>
Create in folder
Create note
Create folder
<<extends>>
Create in category
<<extends>>
Contributor
Owner
Edit note
<<includes>>
Download note
<<includes>>
Preview rendered note
<<extends>>
View contributors of note
Note
Add contributor
Delete
Folder



Data Modelling

We made an ER diagram to visualize our database tables. We used the Lucid charts website for this and the exported pdf with said ER idagram is in the documentation/resources directory of our repo.

Operating Environment and major dependencies

Here's a detailed breakdown of our operating environment and major dependencies:

Frontend

- React.js (v18.2.0): For building our interactive user interface
- Tailwind CSS (v3.3.2): For rapid, utility-first styling
- Marked (v5.0.2): For Markdown rendering
- Socket.io-client (v4.6.1): For real-time communication with the server
- Redux Toolkit (v1.9.5): For state management
- React Router (v6.11.2): For client-side routing
- Framer Motion (v10.12.16): For smooth animations and transitions

Backend

- Node.js (v16.20.0): As our server-side JavaScript runtime
- Express.js (v4.18.2): For building our RESTful API
- PostgreSQL (v14.8): As our primary database
- Prisma (v4.14.1): As our ORM for database operations
- Socket.io (v4.6.1): For handling real-time events
- JSON Web Token (v9.0.0): For secure authentication
- bcrypt (v5.1.0): For password hashing
- zod (v3.21.4): For input validation and type checking

Our application is designed to be deployed on a Linux-based server environment (Ubuntu 22.04 LTS), with containerization using Docker to ensure consistency across development and production environments. We use a microservices architecture to allow for better scalability and easier maintenance.

Authentication

Sign up Process

1. Normal Sign Up
    - Users can enter an email, username and password and sign up with the normal sign up button.
    - Their information is added to the database to be accessed later when they login to the website again.
    - A user's email must be unique, meaning the user is not allowed to sign up with an email that an existing user in the database has.
    - A User's username must be unique.
    - The password must satisfy the conditions:
        - password must be minimium length 6
        - password must contain at least one lowercase letter,one uppercase letter, one number, and one special character
2. Sign up + register passkey
    - Same as normal sign up but user can now register a passkey to be used when they login in again instead of password if they would like to.
    - When a user signs up with this option, the following steps are followed to register the passkey:
        - **Challenge Creation**: A random challenge is generated in the backend/server and sent to the frontend/client. This challenge is used during the passkey registration process to ensure the process is secure.
        - **Passkey Registration**: The client uses the challenge to create a new public-private key pair. The public key and the credentialId is sent back to the server. The server stores the public key in the database, linking it to the user account (the User relation has a publicKey and credentialId attribute).
        - **Storage**:the public key is stored in the database under the user's record. when the registerPasskey function in the backend is called, it receives the credential object and the publicKey. So the fileds like credentialId and publicKey can be used later for authentication.
        - **Use of Public Key**: During subsequent logins, when the user tries to log in with a passkey, the server sends a new challenge. The client signs this challenge using the private key and sends the signed response back. The server then verifies this response using the stored public key to ensure the authentication is legitimate.

Login Process

- Users can enter their email, and password and log in with the normal login in button.
- Loggin in has the same restrictions on password as with the Sign up.
- Users cannot login with a passkey if they have not registered a passkey.
- Users do have the option to register a passkey on the Login page as well if they did not do so when signing up.
- However, they must enter a password to register a passkey - this is to ensure that this is indeed their account for which they want to register a passkey. Otherwise some ill-intended person, knowing a user's email only, could just register a passkey for their account on the login page and use it to log in and gain access.
- remember me is implemented.
- forgot password is implemented, so user can reset their password on the login page.

High-Level Description of Design Patterns

Backend

The backend is built with Node.js using the Express.js framework. Sequelize is used for interacting with the database. The backend has been organised using the Model-View-Controller pattern:

Models: Define our data structure, such as User, Note and Folder entities. Controllers: These handle incoming requests from frontend, such as noteController for any note-related calls. Views: The views in our case are the JSON responses sent to client.

We have implemented a Singleton pattern with Sequelize, so that only a single instance handles the database operations.

Frontend

The frontend is built with React, and we have structed it like so:

Context: We use ThemeContext and UserContext to manage global states like theming and user information, eliminating need for props. Custom Hooks: Hooks such as useTheme and useUser allow us to reuse logic across different components. React Router: This handles our navigation. Framer Motion: We've integrated Framer Motion for animations, which makes interactions more engaging. Responsive Design: We use Tailwind CSS which allows out app looks great on different sized screens.

Contributions

Daniel

I was working mostly on backend/frontend integration. Particularly setting up CRUD operations for folders, notes and users in the controllers, creating their routes and setting up calls to routes in frontend. I also set up the reset password by email link use nodemailer. I set up avatar upload and storage with disk using multer. I created the FolderCollaborators and NoteCollaborators models in order to set up sharing of notes and folders with the AddContributer component. I worked on delete profile (and deleting associated notes and folders). I also set up the Templates as an additional feature. I also researched into patching deprecated issue with the webpackDevServer.config.js in the react-scripts node module and found a solution on github. I helped set up ngrok such as adding header 'ngrok-skip-browser-warning': 'true' to endpoint calls. Used madge https://www.npmjs.com/package/madge (https://www.npmjs.com/package/madge) to help resolve circular dependencies.

https://expressjs.com/en/guide/error-handling.html (https://expressjs.com/en/guide/error-handling.html) - error handling for routes

https://expressjs.com/en/guide/writing-middleware.html (https://expressjs.com/en/guide/writing-middleware.html) - middleware functions

https://sequelize.org/docs/v6/core-concepts/validations-and-constraints/#error-handling (https://sequelize.org/docs/v6/core-concepts/validations-and-constraints/#error-handling) - constraints for sequelize (models)

https://react.dev/learn/conditional-rendering (https://react.dev/learn/conditional-rendering) -Conditional rendering for UI

https://nairihar.medium.com/graceful-shutdown-in-nodejs-2f8f59d1c357 (https://nairihar.medium.com/graceful-shutdown-in-nodejs-2f8f59d1c357) - Helped with graceful shutdown

https://www.dhiwise.com/post/how-to-manage-user-alerts-with-react-notification-component (https://www.dhiwise.com/post/how-to-manage-user-alerts-with-react-notification-component) -Helped with notification for delete user

https://github.com/facebook/create-react-app/issues/12035? (https://github.com/facebook/create-react-app/issues/12035?) We had the same dependency deprecation warning as others in this thread, webpackDevServer.config.js was deprecated in react-scripts node module. Was able to patch it by using recommendation from https://github.com/facebook/create-react-app/issues/12035#issuecomment-2169043485 (https://github.com/facebook/create-react-app/issues/12035#issuecomment-2169043485)

https://dev.to/nsantos16/why-is-ngrok-returning-html-3jn3 (https://dev.to/nsantos16/why-is-ngrok-returning-html-3jn3) This helped sort out ngrok tunnel

Anna

I was working on the frontend development along with Yedida. I created the initial frontend mockup, detailing the UI/UX of the application. Throughout the design process, I ensured that the application was responsive, using media queries to allow for adaption to different screen sizes. I incorporated rendering of the markdown notes using marked in order for the users of the application to preview their work. Most of the first draft of the application was created using only Tailwind

CSS; I worked to incorpoate repeated elements into an external CSS style sheet to maintain consistency, while still keeping the benefits of Tailwind CSS. I used react memo to increase the performance of components that don't require frequenct rendering.

https://logo.com/ (https://logo.com/) - to generate the logo https://www.w3schools.com/react/react_memo.asp (https://www.w3schools.com/react/react_memo.asp) - used to skip rendering if a component has not been changed https://marked.js.org/ (https://marked.js.org/) - for marked rendering

Yedidia

I focused on frontend development, using modern web tools to build a user-friendly and responsive interface. React's core concepts, including hooks like useState and useEffect, managed state across components, while Tailwind CSS ensured a clean, responsive design. I added animations with Framer Motion for smooth transitions and used the Intersection Observer API to optimize performance with lazy loading. Additionally, I managed dark/light mode preferences through React Context API and settings with localStorage,creating a polished and efficient user experience with the help of v0 dev.

Most of the frontend was made making use of the following resources:

https://reactjs.org/docs/getting-started.html (https://reactjs.org/docs/getting-started.html) - Getting the general React concepts, hooks, and component structure, it was used in a lot of cases to implement the state management in most pages|components in our project, we made use of the 'useState' and the 'useEffect' hooks.

https://nextjs.org/docs (https://nextjs.org/docs) - making use of the App Router, server-side rendering and most of the API routing, this would be relevant if we were to implement server-side rendering for the notes list in the Dashboard component.

https://tailwindcss.com/docs (https://tailwindcss.com/docs) - making use of the Tailwind CSS framework for styling components and responsiveness, we used it for instance for styling the buttons in the Note page, such as the "Edit", "Preview", and "Download" buttons.

https://www.framer.com/motion (https://www.framer.com/motion) - making use of this for animations and transitions, Used for implementing the animations in the Note page, such as the transition between edit and preview modes.

https://lucide.dev/guide/packages/lucide-react (https://lucide.dev/guide/packages/lucide-react) This assisted me alot with finding icons for the project, Used for adding icons to the buttons in the Note page, such as the Download icon.

https://reactrouter.com/en/main (https://reactrouter.com/en/main) - The React Router documentation assisted with setting up the routing for our project. Used for setting up the routing in the application, particularly for navigating between the Dashboard and individual sections you can see on the sidebar component.

https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API (https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API) - The lazy loading and scrolling was made making use of the Intersection Observer API. We made use of this implementing some of the lazy loading in the project.

https://reactjs.org/docs/context.html (https://reactjs.org/docs/context.html) - Doing the theme management and switcher using the React Context API was helpful for instance we made use of ThemeContext to make this possible for managing dark/light mode across the application.

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API) Given the theme management and switcher, we needed to keep the theme consistent throughout so this article assisted us with ensuring that was possible, used for storing the user's theme preference in localStorage so it persists across sessions.

https://ui.shadcn.com/ (https://ui.shadcn.com/) This had a couple of pre-built componets we could utilise, which made things easier for us in the implementaton process of the project. We made use of this in the header for instance the drop down menu when we click on the user's profile avatar.

https://v0.dev (https://v0.dev) made use of this AI tool to furhter enhance the UI/UX.

Sinconor

I was working mostly with the backend/frontend integration as well with Daniel. Particularly for the authentication, so the Login and Sign up page. I worked on the Passkey extra feature in addition to normal login and sign up. This meant having to set up endpoints for these features to add the user information to the database etc so that the frontend and backend communicate properly. I also was responsible for ensuring the database tables are in BCNF, which I did by investigating the functional dependencies. Then I also ensured that the user passwords are hashed, using Bcrypt library which salts the password when making the hash for it. Furthermore I worked on the view recently used notes/folders page (set up both frontend and backend for it) and the feature of adding categories and filtering the search using the categories (both frontend and backend). I was also responsible for moving code form the dev branch to main branch on our gitLab repo.

https://web.dev/articles/passkey-registration (https://web.dev/articles/passkey-registration) - I used this to set up challenge generation and getting credentials.

https://medium.com/free-code-camp/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926047c (https://medium.com/free-code-camp/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926047c) - I used this resource to understand how to go about backend/frontend integration.

https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/ (https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/) - I read this article to understand the blowfish algorithm.

https://en.wikipedia.org/wiki/Bcrypt (https://en.wikipedia.org/wiki/Bcrypt) - Then I found out about Bcrypt through this article which explains what it is and how it is based on the blowfish cipher as well as its ability to create a salt for the password.

https://auth0.com/blog/hashing-in-action-understanding-bcrypt/ (https://auth0.com/blog/hashing-in-action-understanding-bcrypt/) - this article showed examples of how to generate a hash using Bcrypt that can be used for password.

https://www.lucidchart.com/pages/ (https://www.lucidchart.com/pages/) - used this webiste to draw the ER diagram for the database.

Pontsho

My contributions covered both UI features and API integration, with a focus on enhancing functionality and ensuring secure user interactions.

1. Frontend/UI Contributions

   Display List of Contributors: Developed the feature to display who is currently editing or viewing a note, improving real-time collaboration visibility. Markdown Functionality: Added Markdown support, enabling users to format notes with live preview functionality.

2. API/Frontend Integration

   User CRUD Operations: Worked with Daniel, who handled the core CRUD operations. My role was to implement authentication and authorization using JWT, ensuring that only authenticated users could access and manage their accounts. Categories Authentication/Authorization: Set up authentication and authorization for categories, securing the creation, update, and deletion of categories for authorized users only.

3. Database Security

   Sanitizing Database Queries: Secured the API by sanitizing database inputs to prevent SQL injection attacks, using tools like express-validator.

4. Real-Time Collaboration with WebSockets

   I set up the foundation for real-time collaboration using WebSockets, allowing multiple users to edit notes simultaneously. However, achieving live updates without requiring page reloads has been challenging, and this feature is still being refined to work seamlessly with real-time note updates and conflict resolution.

5. Conclusion

   My contributions included implementing secure authentication, Markdown functionality, and initial real-time collaboration features. Despite challenges with live updates via WebSockets, significant progress was made in delivering a collaborative platform with a focus on security and user experience, alongside working with Daniel on CRUD operations.

Resources

https://socket.io/how-to/use-with-react (https://socket.io/how-to/use-with-react) - Learn a practice example on how to use sockets in react web app and studied a basic implementation https://socket.io/get-started/chat (https://socket.io/get-started/chat) -Why use sockets and How to use sockets to listen in the server.

https://medium.com/@novaliastevany7/sequelize-model-for-user-authentication-in-node-js-605355aee491 (https://medium.com/@novaliastevany7/sequelize-model-for-user-authentication-in-node-js-605355aee491) - Article onSequelize model forUser Authentication Node.js https://stackoverflow.com/questions/6711313/database-storing-multiple-types-of-data-but-need-unique-ids-globally (https://stackoverflow.com/questions/6711313/database-storing-multiple-types-of-data-but-need-unique-ids-globally) - Database storing and multiple types of data

https://stackoverflow.com/questions/21694933/how-can-i-get-a-specific-data-using-sql-select-query (https://stackoverflow.com/questions/21694933/how-can-i-get-a-specific-data-using-sql-select-query) Got Guidelines on "How can I get a specific data using SQL select query"

https://www.imperva.com/learn/data-security/data-sanitization/ (https://www.imperva.com/learn/data-security/data-sanitization/) Article on What Data sanitization is/

Additional Features

We implemented Passkey support for login, as well as the "Lost passwords can be reset via email and email notification when a note is shared with you" additional feature. For our own ideas, we implemented templates for notes as well as the ability to download notes. We also patched a deprecation issue within the react-scripts node module, but we did not make this patch ourselves, see https://github.com/facebook/create-react-app/issues/12035#issuecomment-2169043485 (https://github.com/facebook/create-react-app/issues/12035#issuecomment-2169043485) for patch.

Known Issues

Real-time updating of note/folder

When you have a note open and a collaborator changes a note, a page refresh is needed to see update to note. Other refreshes may be needed in specific circumstances, such as when you are on the "Shared Folders" page and someone shares a folder with you.

Sockets

Live updates using websockets are not working in their current state, there are "Connection Refused" errors.

Avatars

The avatars seemed to be working when running local host, but when using ngrok it was discovered that avatars are not being accessed correctly.

Warnings at compilation

'WARNING in [eslint] src/components/AddContributor.js Line 37:29: React Hook useCallback received a function whose dependencies are unknown. Pass an inline function instead react-hooks/exhaustive-deps'

appears when starting frontend