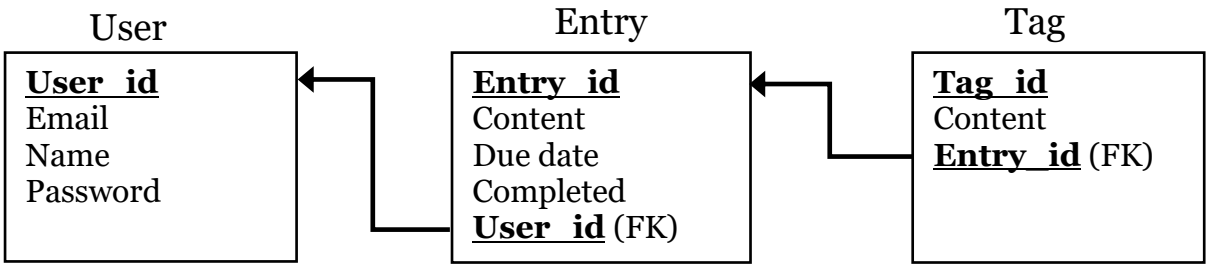
CVWO2020 Toh Zhen Yu, Nicholas A0201406Y

Schema



Features

Redirect with message (eg unauthenticated user)

Search within entry content and tags simultaneously with live filtering

Front end tracks a model of the data, hence no need to reload after update/delete

Mark entry as done without deleting (separate option to delete permanently with confirmation pop-up)

Hosted on netlify/heroku

User login system (buggy, works locally but not when hosted)

Design decisions

I had originally considered a schema where tags were directly under users, and where taggings would be created to link an entry to a tag. This way, in the backend each tag would have many entry through taggings. However, pulling up all entries tagged this way would require much more computation on the back end. Instead, I decided (with my limited understanding of development practices) that this should instead be done on the front end, because one backend server can be serving many users. Hence, I implemented a system where each tag belongs to only one entry, but the front end would filter entries live through a flexible search system. In this way, the backend would simply send the entries to the user, who would process the data on their side.

Similarly, I implemented the front end such that it tracks a model of the backend, and hence does not require a refresh (and query of all the user's data) every time an edit is made. For example, when the user adds a tag to an entry and a successful response is received from the server, the front end updates its display to reflect the change without refreshing the page.

Some todo managers implement a system where entries can be put into categories. However, I chose not to do this, as firstly this can already be done easily using tags, and secondly tags are even more powerful, as in most implementations (such as GTasks) an entry can only be put into one category, whereas an entry can have multiple tags. Users can also filter by multiple tags simultaneously (see readme).

I made the site as intuitive as possible, hence I feel that a user manual is unnecessary, though there is a visual guide on the github readme.

Areas for improvement

I used css for designing the front end page, as I could not fix some node-sass errors that resulted from using bulma's advanced libraries features. I would like to try bootstrap or material UI, as the prebuilt framework would likely be more visually pleasing than whatever I could do manually, while also saving time.

The code is mostly dry, except for the various axios calls to the API. I would separate out the axios settings into another file and import it into the various pages, where the code for making an API call and displaying the error message would be much shorter.

I could implement a tag autocomplete system (in the add tag input field) that suggests tags from a user's other entries.

Currently, users can only add tags to an entry through its edit page after the entry has been created.

I would want to learn and implement typescript and redux. For the scope of this project, I feel that typescript was unnecessary. Redux would be useful for tracking a model of the data across the application.

For user authentication, I spent a huge amount of time on devise and later switched to using session, but neither worked. I would instead try explicitly handling the JWT.

Difficulties with user authentication system

I originally intended to do authentication using devise but it could not work. I then separated the front end from the API, and switched to using session. This worked in development and also when running the production build locally, but for some reason does not work when hosted. I narrowed the issue down to the front end

sending a blank JWT, but could not fix the bug. I had posted on stack overflow on 10th Jan but have not found a solution. The code uploaded to github is the original buggy code, however for the purposes of demonstration I had hardcoded the user in current_user_concern.rb.

My Thoughts

I think it's useful to have learned react and rails as a result of doing this assignment. I even used these tools to build a web application for a recent hackathon. Some people told me that rails is not the most cutting edge database system and may not scale well or be suitable for complex applications, though rails has a proven history and an extensive library of gems. As for this assignment, since most students, myself included, have had no experience in web development, database design, or ruby, I felt that the learning curve was very steep, though the difficulty was ultimately palatable.

Ruby as a language was itself challenging and unfamiliar. One example is that in the languages taught in 1K CS mods, it is simple to pass a function into another function. However, in ruby, methods are called even without parentheses, and hence one has to deal with symbols and other complexity to pass methods around. Thankfully, javascript was much more similar to language paradigms I am familiar with. As a sidenote, I was fascinated by conditional rendering in react, it seems like a very elegant hack.

Another intimidating factor was needing to use systems I did not understand, such as WSL and webpacker, especially when there were issues with them.

In total, I think that the difficulty and open-ended nature of the assignment is beneficial to CVWO team as it filters out students who did not persevere through the challenge. As a student, however, I would have preferred a more structured assignment, for example if we were given directions to use JWT or a certain styling framework.