What I feel about my accomplishment:
In short, I have learned a lot from this assignment. Given the time constraints, I have tried my best to come up with the website with its major features and necessities.

Tech stacks and libraries used:

1. React.js (with hooks)
2. Typescript
3. Redux
4. Ruby on Rails (for REST API)
5. PostgreSQL
6. Netlify (for frontend deploy)
7. Heroku (for backend deploy)
8. Heroku Postgres' addon to automatically backup database under a certain schedule
9. JWT (for authorization)
10. Search autocomplete

Features:

1. Create todo with 1 or more tags
2. Edit todos
3. Delete todo
4. Mark todo as done
5. Search by tags (with autocomplete)

Design considerations:

1. JWT token
    - I did not specify an expiry time of a JWT token since this app would not have security issues if I do not make it expired after a certain amount of time
    - The JWT token is stored in the cookies instead of localStorage, with reasons specified in the "I learned more about client-side storage" section
2. Schema
    - User has many tags and todos
      Since one user's tags and todos should not be associated with other user, I design it such that a user has many tags and todos
    - Todos and tags through taggings table
      Since a todo might be related to multiple tags, and a tag might have multiple todos, I used a taggings table to record the association of todos and tags
3. Auto delete tag when deleting the last todo associated with a tag is deleted

- The user do not need to manually delete a tag

My key takeaways:

1. Create the database schema early since it might determine the flow of the development or design of the program
   When creating this todo app, I created the todos functionalities first (create todo and tagging). By the time I wanted to start doing the user-related features, I realized that it might be better to create the relation between user and (tags and todos) to be one-to-many. Hence, I needed to change the existing schema and change some of the REST API design and implementations
2. Typescript greatly improves the scalability and minimizes error
   I am able to appreciate how typescript saves a lot of time by showing errors as early as possible, which is during compile time. I can also see the benefits in the Intellisense (such as code completion) which boosts my productivity.
3. React hooks simplifies the code
   I feel that my code quality has improved greatly by using React hooks, as opposed to class-based React. This includes a much shorter code and consistency in the style of code. One example of it is that instead of using the various functions (componentDidMount, componentDidUpdate, etc), we can use useEffect.
4. Commit frequently
   During the project I was able to try various sorts of things--experimenting with external libraries, solutions from some tutorials or stackoverflow, etc--without worrying it might mess up my code.
5. Git Branches simplifies deployment
   Using Heroku and Netlify, I can specify automatic deployment on a certain branch. By doing so, I can deploy a working iteration that might have different configurations than that of the master branch (or main branch, in my case).
6. The life-cycle of development
   Since I have targeted myself to be able to deploy this app, I tried deploying this app early, despite the fact that there are still a lot of unimplemented features. This reduces the potential challenges that might come after the codebase is too large to debug on why the deployment fails.
7. I learned more about client-side storage
   As authorization requires some data to be stored in the client side, I learned the differences between cookies, localStorage, and sessionStorage. At last, I used cookies since it is automatically included in every request, more secure (by enabling http-only, the client's browser cannot directly access the items via console), and that the token size is still appropriate to be stored in the cookies.

8. Pay attention to the trailing slashes in urls
9. Be aware of the small details of the framework
   In this case, it is that Rails have conventions depending on the contexts, such as plural for controller (users), singular and capitalized for model (User), plural in has_many (users), singular in belongs_to (user)

What could be improved:

1. Use Docker
   Currently continuing this project in another device would still be simple (install the Ruby gems and npm dependencies for frontend). However, for a larger scale, it might reduce the pain of differences in each computer's environment.
2. User interface
   Although I believe that the website has an intuitive and simple user interface, there might be a better color schema.
3. Verification through email