What I have achieved:
Website: https://memoapp-bryan98.netlify.app/
API: https://memoapp-bryan98.herokuapp.com/

Deeper understanding on principles of APIs:
Most of my development work in the past involved making API calls, which at the time I found very frustrating when I couldn't get what I wanted easily, and instantly deemed backend development a trivial development work. After developing an API-only application on Rails, I was exposed to the importance of limiting certain interactions between the backend and the developers due to security concerns, backend design, DRY principle (don't develop something that can be easily derived, keep it lean!), etc. Developing APIs is no easy feat, and even if the end result seemed simple, the thought process that a developer went through to reach that conclusion was long and arduous!

It is important for a backend developer to think fast and thoroughly! When I was developing, I was lazy in the beginning, and decided to return everything in the database to the user. Not only was this an inefficient design choice, as the amount of rows increased, the front-end would also slow down due to waiting for overwriting data that was unchanged. As a result, it may be better to just return one entry, and leave the responsibility of updating the front-end with newly added data to the developer, be it to concatenate the new data to the array/JSON containing previous data, or to do something that the developer deem necessary.

All in all, this experience opened my eyes to backend development, and I definitely will consider doing backend development again!

Created a React front-end that interacts with Rails
In the past, I have read up on what Angular.js, React.js, etc. are at a high level. Despite that, actually developing with a framework feels completely different from just "knowing" it.

My approach for the front-end was to "Just make something usable". I focused on making it functional first, before focusing on styling and choosing better looking UI elements. In fact, most of my initial UI component choices were not altered. I did my best in segregating components, such as Memos having multiple Memo objects as well as access to categories, each Memo object had to be clickable, to allow a form to pop up in place. Although the code is nowhere near "easy to maintain", I think it's fairly straightforward to trace where each property/function comes from. I separated the property type from the main code, thinking it would be easy to keep track . It's probably subjective to me as I usually have 2 windows open, the right side is references to models, and the left side being the actual code.

I made a Pomodoro timer, as I really wanted to try and make one in JS. Initially, I implemented it as a separate object, and tried to integrate into the component. Due to lack of time I decided to couple them together, which might seem like a bad decision, but realistically speaking the timer didn't need to be separate from the component given the scale of the project and the time left to implement one. At the very least it works like a Pomodoro

Timer, with every three sessions leading to a longer break, which I have grown to like using it. At the time of writing this report, I have set all the timers to be 5 seconds to see it working; I might add a developer version to toggle between real usage (5 sec ver vs 25 min ver),

Initially, I use class based components for development. I noticed how verbose it was, and I felt there shouldn't be a need to be that verbose, and using Hooks might help alleviate that problem by allowing React library to handle the objects for me. Furthermore, the documentations for Materials-UI, which was the front-end UI library I've decided to explore, was using Hooks completely. The two reasons mentioned compounds with each other as well, as I had to understand how Hooks worked to translate it to the class version of React in order to use Material-UI. The verboseness of the code meant that my code was difficult to maintain as well. I made the hard decision to refactor all my code to Hook based React.

Familiarized with Typescript

The most I had done with JavaScript (excluding Source) was writing a NodeJS server, which I didn't do a great job at either. I remember it was tiring, how I had to keep track of the various variables available, as well as read up on all the docs on what is available. Although I still needed to do that for TypeScript, the difference comes when I had to write out what fields are available for a specific argument. This allowed some nifty features like error checking and autocompletion, which meant that I was confident with the correctness of function/attribute access, and only needed to focus on whether my logic was 100% correct.

TypeScript forces me as a developer to know what I am actually doing with my code. For instance, writing a style object for Material-UI felt like just passing a JSON as a property. When I finally realized what the squiggly lines meant, I was shocked at how shallow my understanding in the usage was, the object of CSSProperties for Material-UI can be represented as a JSON object, and I had to specify even that!

Furthermore, TypeScript ensured I was consistent throughout what I categorized as the same object. For instance, I initially thought a new Memo object should be exactly the same as a New Memo object. As I developed, I drew a clear line between them, as a MemoForm object need not have an ID, but all Memo objects have to have an ID. The same goes for Categories, as well as Memoboards. Something I could have done better would be to make what they had in common as interface, and extend the interface as a new type based on what MemoForm and Memo required. By the time I noticed this, I was too far down the rabbit hole. This is something I will work on, identifying what is common between object types and abstracting them.

User Manual:

1. You will first be introduced to some credits to authors of images as I have included an Icon from Flaticon. This will vanish relatively quickly as the app loads up.
2. You will be greeted with a Welcome page. This Welcome page is intended to have user sign ups and logins, as I would want others to use this as a tool for themselves someday. Click on the link to continue.

3. You will then be greeted with an Overview (previously called Master Memoboard), containing all your memos from all memoboards. Click on the Memoboards name to switch to a Memoboards.
    a. Clicking on Add Memoboards allows users to create a Memoboard. For now, each Memo belongs to a Memoboard, and cannot be moved to a different Memoboard, so do be aware of where you create Memos!
    b. Omit the word "Memoboard" when naming your Memoboard. Note that there is no way to edit Memoboard name from the UI for now.
4. Click on "Add a Memo", and a Card based component will appear. Add in some title text and body text, and the memo can be created.
    a. Notice the three dots beside the button "Cancel", you can assign categories for the Memo during creation. Go ahead and click on the words to assign them.
    b. It is advised to not click "Add Categories.." during Memo creation. There is no bug, just that the page will refresh, losing the current edits.
5. You will be greeted with some memos. Notice the Memos are not in a traditional grid. This is intentional, as using Masonry allows Memos to "fill up" vertical empty spaces.
6. Click on any memo, and the Memo will be editable in a form. Do note that this is the only place to copy the body/title of the Memos.
    a. Remember to click "Update"! Changes will not be saved otherwise.
    b. Clicking on another Memo will close the current Editing Memo, and make that Memo editable instead.
7. You can create Categories. Click on any Memo/Add Memo, then click on the three dots, and you will see at the very end there is an option to "Add Categories…".
    a. Clicking on it reveals a Color Box, a separate library from Material-UI. In this Color Box, you can pick different colors, and change the category name. However, there seems to be some bugs with editing the hex values directly. Use the Color Picker or RGB value instead!
    b. Note that creating a category refreshes the page. If you were editing a Memo beforehand, click cancel, and save your Memo before creating a Category.
8. There is a "filter by…" under the Memoboard Name and above the "Add a memo…" box, clicking that reveals a drop down to filter by Categories.
    a. Click on a category, you will see categories in a circular box, called Chips in Material-UI. Currently the only way to remove this filter is to manually deselect them from the list.
9. On the bottom right, there is a Floating Action Button (FAB). Clicking on it reveals a Pomodoro timer. At the time of writing it is configured to run for 5 seconds regardless of which stage it should be at. This is for demonstration purposes.
    a. I will update it with a showcase version and a developer version. However, it may not reflect in the Github submission.
    b. When the Timer ends, you will hear a ringing. You can stop the ringing by
        i. Clicking on a tab in the Pomodoro Timer
        ii. Clicking "Start Timer" Button
        iii. Clicking outside the Pomodoro Timer.