





CVWO Final Writeup

Name: Tan Zong Zhi, Shaun (A0235143N)



My Journey

I started out with only a little experience in React and git.

During the development of this app, I learnt and used:

-  Git
-  Ruby on Rails (with Gem – devise_token_auth)
-  React (with Component Library – MUI)
-  Redux (with Middleware – Redux Thunk)

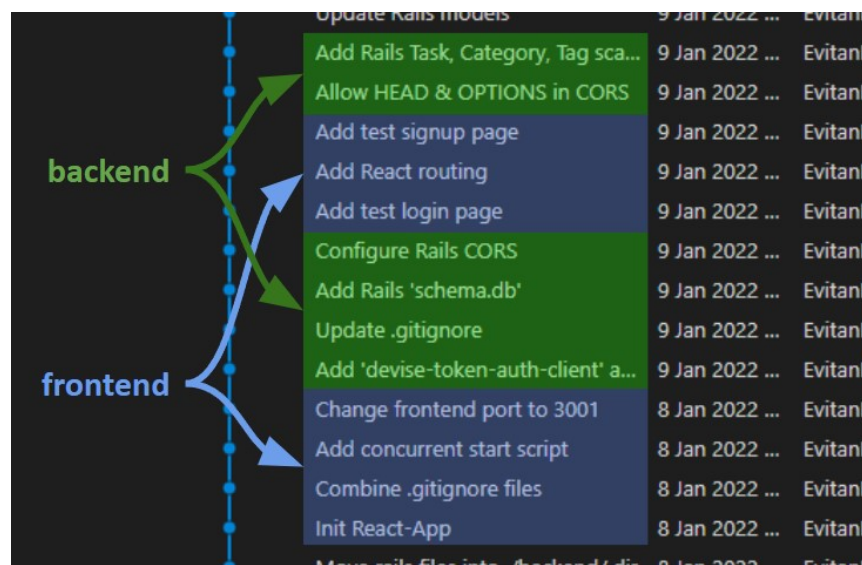
along with some tools for testing / prototyping:

-  Postman (to test my API)
-  Figma (to prototype frontend styling)

Things I learnt

1. Separate backend and frontend into 2 repositories

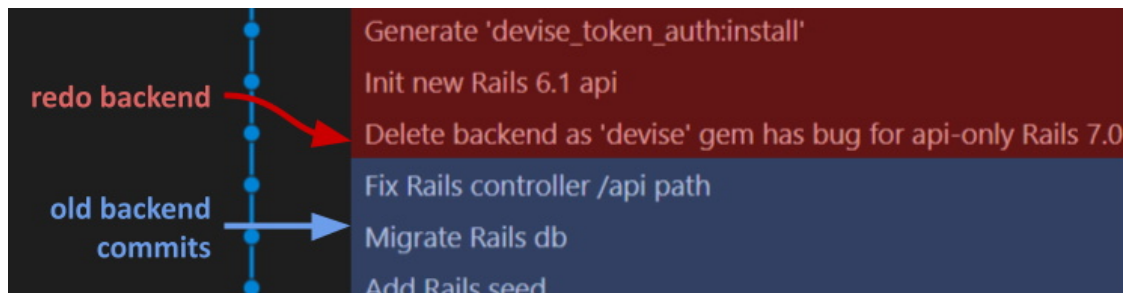
When something goes wrong in the backend, it was hard to find the correct backend commit to git checkout to when they're all sandwiched between frontend commits.



2. Don't always use latest versions

I had to redo my backend twice due to incompatibilities between the latest versions of Rails, Ruby and the Rails gem – devise.

- **Rails 7.0.0** incompatible with **Ruby 3.1.0**
(<https://github.com/rails/rails/issues/43998>)
- **Rails 7.0.0** API-only App incompatible with **Devise 4.8.1**
(<https://github.com/heartcombo/devise/issues/5443>)



Moreover, using the latest versions meant a lot of older online tutorials were too outdated to follow.

- Rails gem `simple_token_authentication` mentioned in a YouTube Rails API Tutorial didn't support Rails 7
(https://github.com/gonzalo-bulnes/simple_token_authentication/issues/385)

3. Look for libraries / resources that have already done what you need

I tried creating the React frontend from scratch via `yarn add react react-dom`, before realising I needed a bundler like Webpack.

Then I spend an entire day configuring Webpack with the required loaders and plugins (eg. `css-loader`, `html-webpack-plugin`), failing to get it working, and got frustrated.

Before finally, starting over with `yarn create react-app`.

Typescript

It's my first time using Typescript, and coming from a mostly Javascript (dynamically typed) background, it was initially quite hard to transition and get used to.

But by the end, I found it really helpful, especially when renaming and changing functions. It shows me where/how things break immediately when I make a radical change, rather than having to find out 5 commits later.

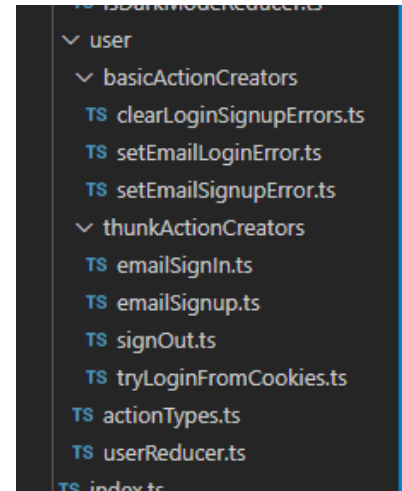
It also helped a lot how Typescript shows me the parameter types that libraries expect me to give, rather than having to check the documentation every 10s.

Redux

Definitely a lot of boilerplate for simple things.

The store for my user object alone consists of 9 files. And I almost have as many files for redux as I have for the rest of my frontend.

But it's definitely nice to not have to keep lifting the state up, and end up with tons of props on the container component.

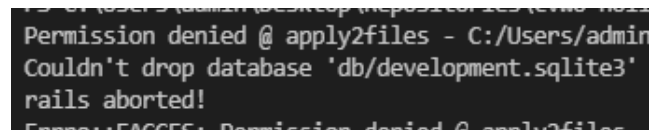


Ruby on Rails

Rails is by far the most difficult of all the tools to learn. Sometimes, documentations (especially ones geared towards beginners) are rather lacking.

Sometimes Rails names their variables/files in camelCase, other times in hyphen-case.

Also, "rails db:reset" doesn't work on Windows. (<https://github.com/rails/rails/issues/31589>)



Coupled with the various bugs and incompatibility between the latest versions of Ruby, Rails and its gem – Devise, it was really difficult for me to grasp.

Responsive Design

A large part of my time was dedicated to making the app responsive and mobile-friendly, so much so that I had to drop a number of features (like Google Calendar integration) due to lack of time.

It was a lot more work than I anticipated; using Grid to fit a varying number of tasks in a row depending on the viewport, hiding the sidebar on mobile, implementing long-press for mobile to mimic the double-click edit task/category functionality, etc.

But I really believed that for a todo web app (or any web app really) to be relevant, it has to work on mobile.

