INTEGRUM ESG WRITE UP

I've written a summary of the changes made, and some commentary about each task. Hope this is sufficient!

DARK THEME

- Had to learn TypeScript (mostly the same as React)
- Had to learn Tailwind

Additions:

- Button on top bar to change themes
- Extracted colours into default theme
- Added variables for theming
- Added dark mode theme

Challenges:

- Typing for passing props
- Learning tailwind
- Getting theme change to register (ended up using document.setAttribute)

For the dark theme, probably the biggest hurdle was the initial shock of working out how Tailwind works, particularly for theming as I previously used SASS for that. I ran into a brief hiccup where the strong typing made it harder to pass what I wanted to child components, but got that all working. Initially I just had a toggle button at the bottom of the page, but wanted to move that into the bar component while leaving the "theme" useState in the App.tsx (hence the prop issues). From there it was using what I'd learned about Tailwind to create a "default theme" based on the pre-existing colour scheme from the app, and then making an alternate greyscale "dark" theme, inspired by Discord's palette. The last thing was getting the actual change to apply to all elements, which I did by using a document attribute, a technique that I carried over from previous projects.

SCHEDULED DATE

- Had to learn Prisma

Additions:

- Add scheduled date
  - Second date on frontend (input)
  - Tweak data being sent back and forth
  - Second date on database
  - Second date display
- Add time left / time until for tasks

Challenges:

  - Learning Prisma (figuring out how to update schema)
  - Keeping timer up to date constantly (using setInterval as a timer)

This one I found a lot easier than the dark theme. The only tricky part was finding out how to update the database schema. From there it was just adding the functions on the front and backend which didn't take very long. I considered having the additional datetime automatically register as the current time (aka a new date object) but decided against it for the sake of functionality. If the user wanted to set a task in the future, this would not be possible with the automatic idea – same with if the user wanted to set the beginning of a task in the past and the end in the future. An extra little thing that I thought would be cool was to add a little timer for each task to show how much time left they have, either remaining or until start – or if they're overdue. I placed these in the top right of each task widget, opposite the title.

EXTRA TWEAKS

Additions:

  - Added redirect to main menu after adding/editing tasks
  - Stopped dates overflowing
  - Stopped titles and descriptions overflowing by truncating
  - Added cookies to preserve theme between sessions

Challenges:

  - CSS formatting

I know these weren't asked for, but as I was messing with the application I noticed some text elements overflowing their <div/>s. I also just thought that having it redirect back to the task list felt better than having it stuck on the create/edit pages. Another uncalled-for but sort of useful addition was adding a cookie to store the theme. In the context of a real application there's complications with consent laws with regards to cookies, but I decided to add it here as that's not really applicable on an assessment task.

TESTING

- Had to learn testing suite

Additions:

  - Singular very long test function to consecutively test each function

Challenges:

  - Learning testing format
  - Dealing with jest running test functions concurrently

This was probably the hardest one. Having not used testing suites for a TypeScript backend before it took some time to fully understand what was going on in the documentation. Once I got the first injection working for the "get all entries" GET request, the rest followed suit rather quickly afterwards. The issue from then was dealing with what appeared to be unpredictable errors being thrown, but which rather was the result of functions being called in parallel, resulting in data races for the database. I solved this by just shoving it all into one consecutive function, designed for function coverage of the backend. I could have gone for line coverage by testing the alternate branches of the functions with bad inputs – but firstly, I felt it would be excessive for the sake of the test; and secondly, because of the parallel function calls (and me failing to figure out how to apply the Jest "--runInBand" to the NPM test script) it would have needed to be one ludicrously long function, which would have been utterly diabolical.