# Integrum ESG - Card App

Lucas Mockeridge

## 1 Dark Mode

The first part of implementing the dark mode feature was to add a 'Settings' button to the navigation bar. Once clicked, this button opens a menu containing a checkbox, through which the user can turn dark mode on or off. Since the 'Settings' menu was implemented with a `<dialog>` element, the menu can be opened and closed without React needing to re-render the `Settings` component. Therefore, all that is required is to move between the `<dialog>` element's open and closed states using the `useRef` hook.

However, React must re-render each component whose colours are altered once dark mode is switched on. Hence, the `useState` hook is essential to the implementation of the dark mode feature. Whether dark mode is enabled or not is captured by the boolean state variable `isDarkMode`, the value of which can be updated using the `setIsDarkMode` function.

Both `isDarkMode` and `setIsDarkMode` are instantiated in `globalContext.tsx`. Ostensibly, the purpose of this is to provide the `Settings` component with the `toggleDarkMode` function while hiding unnecessary implementation details. But the real purpose is to share `isDarkMode` and `toggleDarkMode` throughout the component tree without the hassle of passing props. This is achieved by creating a `DarkModeContext`, which represents the two elements, and using a `DarkModeProvider` to allow components to subscribe to changes in the context through the `useContext` hook. An added benefit of this arrangement is that the logic behind each of the application's contexts is located in `globalContext.tsx`.

The `Settings` component uses `isDarkMode` to determine whether the checkbox should be checked or not and `toggleDarkMode` to alter the application's theme when the user clicks the dark mode checkbox. `toggleDarkMode` produces the dark mode effect by adding a new CSS class to the document's body. When dark mode is turned off, this class is removed. In any case, the user's preference is written to local storage, so that it persists across visits to the application.

## 2 Scheduled Field

Adding a scheduled field to the database merely required altering the database's schema in `schema.prisma`. The only other modification on the backend was to ensure that the scheduled field was set to the current time if it was not provided by the client. The frontend also had to be modified to allow the user to set the scheduled field, and to include the scheduled field in API calls to the backend.

## 3 Backend Tests

The backend tests rely heavily on the `server.inject()` method to make real API calls. This style of testing was chosen in preference to mocking the Prisma client, as the former provides a more realistic simulation of the application's behaviour than the latter. However, using `server.inject()` does make the tests slower, but this was deemed acceptable since they are very few in number.