Sanctuary App

Overview:

The Sanctuary Application begins with a login page where users must input their credentials, which are then validated against the data in mocks.js. Upon successful validation, users are directed to the Dashboard, presenting three tabs: **Pets**, **Adopters**, and **Best Match**.

If the user selects the Pets tab, a list of pets along with their details is displayed in card format. Users are granted the ability to Add, Delete, and Edit pet entries.

Selecting the Adopters tab presents a list of adopters and their preferences.

Choosing the Best Match tab provides users with a list of pets best suited to their preferences, including a matching score and preferences. Upon selecting a pet for adoption by clicking the "Accept" button, the corresponding adopter and pet entries are removed from the list.

Functional description:

Data: Entire application's data is stored in the file mocks.js. As we currently don't have backend support

State management:

1) **Redux**: Maintains list of pets and adopters stored in the store.
   Reason: If animals or adopters are manipulated anywhere in the application it should reflect in all the places. In current application it can get manipulated from Pets and Best Match, and both the component should be in the sync
2) **Context**: It has logged-in user information
   Reason: This can be stored in redux as well as context I preferred context over redux because once the user is logged-in it will never change. We will need this information only if we have to track the activity of the user or want something to be role based
3) **Local state**: Rest are the local states which only impacts the component itself or the child component.

Higher Order Component (HOC): Utilized for displaying Add/Edit form in the modal which is in the HOC

Reason: Modal's design, open/close functionality is common, only the content of the modal will change. To enhance the reusability and consistency in the application.

Common components: Includes InputField, Buttons, NavTabs.

Reason: To increase the maintainability, reusability and consistency of an application.

For example, input fields should be consistent in the entire application only passing the props should suffice. In large applications this helps a lot

Custom hook: Developed to manage input field change events, so that the state change will take place only from 1 file. This makes our code look clean and less error prone.

Constants.js: Centralizes button, field, and header labels to streamline application-wide changes, minimizing debugging efforts and errors.

Technical significance of a component:

The root file of the project is **App.js** where I have added a context for "logged in user". I have added the **Routing** to the login and the dashboard page. Inside dashboard I have added a nesting routing for Pets, Adopters and Best Match pages.

I am wrapping all my components with **Context API** and **Redux** store so that all components have access to the store and context value.

Login Page: Features a simple form with 2 input fields for username and password. Upon successful validation and triggering 'Login' button and user is navigated to Dashboard, however if the user is invalid, he will get a **validation** saying 'Invalid User'.

Dashboard is divided into 2 modules View and Best Match (as per the assign doc).

View part has Pets and Adopters

Pets: It has the list of all the pets in the card format with the details of pets. Each pet details can be edited/deleted. User can also add a new pet to the list. Since the pet count can be more than 1000 records. For the large data-set **pagination** is introduced here. Lazy loading is an alternative pagination. In real scenario where the API call is involved, we can also lazy load the component.

Adopters: In this adopters list is displayed dynamically

Best Match: Displays pets best suited to each adopter based on pet preferences and personality matching. The score is based on 2 conditions.

a. If the adopters pet preference matches the pet. For example, adopter is willing to adopt dog and cat, the dog and pet will be the good match for the adopter
b. Personality preference is the 2nd parameter, we storing personality of each pet in the pet's details and preferred personality for each adopter.

By calculating the overall percentage, we are generating the score out of 10 between animal and adopter. We are excluding the records with 0 score. For all the calculations we have created utilis.js file and using **useMemo** to memorize the calculation. So, if the animals and adopters doesn't change and still the components re-render the useMemo will no recalculate however it will return the memorize value. This helps in increasing the performance.

Further considerations:

1. Styling
2. Unit test
3. Typescript
4. Create a separate routing component instead of writing everything in App.js
5. Form validations
6. Material UI to have accessibility
7. Internationalization (i18n)
8. After accepting we can display a modal showing which adopter and pet got deleted.
9. Hooks folder can be outside common, since it has custom hook related to inputField I have placed that in common.
10. Somewhere the alignment in bit off that could be corrected.
11. We can add more data