Amalgama – Challenge

Ejercicio 1 – Componentes

1. Enunciar todos problemas o posibilidades de mejoras para este componente. Mencionar cuales de los problemas o posibilidades de mejoras enunciados son los más relevantes.

En dicho componente se encontraron varios problemas o posibilidades para mejorar los cuales son:

Problemas principales:

- ***Separación de 'concerns'***: Como primer punto a destacar, se ve que el componente está cargado con demasiada lógica lo cual puede llevar a que la solución, al escalar, se vuelva menos mantenible e ilegible. Este punto también lleva al siguiente…
- ***Diseño de Componente para Contacto***: Para hacer el código más mantenible, es mejor que el renderizado del contacto en sí lo haga un componente propio que se encargue de eso mismo, hacer el display del contacto. Mientras que su padre hace la lógica para 'formatear' la data que llega como prop.
- ***Keys faltantes al realizar los maps***: en React es fundamental asignar una key a los componentes que son mapeados manteniendo una identidad estable en caso de que cambien los ítems mapeados.

Otras mejoras y observaciones

- ***Función para el formateo de data:*** con el fin de que el código del componente no quede tan cargado como se puede apreciar a primera vista, se puede separar dicha lógica y que una función específica haga eso.
- ***Error semántico en las listas:*** los elementos <ul> no deben contener otros <ul> como elementos de la lista, sino
- ***Agregado de PropTypes para el tipado de props:*** Para mejorar la mantenibilidad del código se recomienda utilizar la librería PropTypes la cual permite verificar los tipos de los props que le pasemos a nuestros componentes, y en el caso de que sean incorrectos, informarnos de dicho error.
- ***Utilización de React Memo:*** con el fin de evitar renderizados innecesarios, y más si la cantidad de contactos crece, se propone la utilización de React.Memo tanto para los contactos mapeados como para el componente en sí.

2. Refactorizar el código y adjuntar cómo quedaría la solución luego de la refactorización.

https://github.com/gsanchezzusaeta/amalgama/blob/main/src/components/Contacts/ContactsScreen/ContactsScreen.jsx

3. Justificar lo realizado en el punto 2 explicando qué mejoras aporta y por qué soluciona lo comentado en el punto 1.

- ***Se separan los 'concerns' y mejor legibilidad:*** Primero se abstrae la lógica de formateo de data de la constante contactsToDisplay, dejandola en un archivo utils del componente. También, para la modularidad de la estructura, se separa la información de cada uno de los contactos en un componente aparte cuya función será la de realizar el display de los mismos, mientras que el componente ContactsScreen solo recibe la información y la mapea.
- ***Agregado de keys en maps:*** Para la mejora del rendimiento y la identificación por parte de React de cada componente, se agrega una key a cada componente mapeado, también evitando advertencias en la consola :)
- ***Agregado de PropTypes:*** Para realizar la validación del tipo de cada uno los props de los componentes, se utiliza la librería PropTypes, lo cual mejora el troubleshooting de errores y más si el código escala.
- ***Cambio semántico de ul:*** Se agrega un elemento <li> el cual el <ul> de las direcciones del contacto.
- ***Se agrega Memo:*** como se explico antes, se agrega memo tanto al componente como para a la constante de contactos a mostrar, acá haciendo provecho del hook useMemo.
- ***Mapeo en data de Address:*** Para reducir cantidad de código se optó por realizar un mapeo de las entradas de address ya que renderizo todas las que tiene.

4. Se pide agregar una vista de perfil del contacto (layout similar a como se muestra en la lista), suponiendo que los datos del contacto son avatar , first_name , last_name , company , details , email , phone_number y address . Adjuntar la solución propuesta.

Reutilizo el Componente creado anteriormente

https://github.com/gsanchezzusaeta/amalgama/blob/main/src/components/Contacts/ContactCard/ContactCard.jsx

Ejercicio 2 – Estados

Para el manejo de la información en la aplicación se optó por utilizar Redux Toolkit (RTK), la cual provee una forma concisa y optimizada de realizar el manejo de la aplicación entre muchas otras cosas. RTK también cuenta con una funcionalidad que permite guardar los 'slices' en el localStorage, por ejemplo.

Para su implementación es necesario crear los archivos que se encuentran bajo la carpeta redux en el repo:

https://github.com/gsanchezzusaeta/amalgama/tree/main/src/redux

Ahí tendremos una carpeta para nuestros **reducers**, un archivo para el **store** y otro para el provider el cual encerrará nuestro App.js

Primero establecemos nuestros **reducers**, el cual tendrá el objeto inicial de mis estados y las distintas funciones que querramos ejecutar para cambiar nuestros estados.

En nuestro archivo **store**, crearemos nuestro almacenamiento, como bien dice la palabra :) . Definiremos la configuración del mismo, los reducers que utilizará, middleware si es necesario y también si queremos persistir nuestros estados.
En nuestro authPersistConfig aclararemos que estados quiero persistir y cuales no. Yo aplique la persitencia de mi data como para mostrar dicha funcionalidad, en el caso del token lo voy a querer persistir. En el caso de las respuestas que recibo por API del ejercicio la manera en que se planteo guardarlas fue la siguiente:

```
{
  "books": {
    "1": {
      "id": 1,
      "title": "Clean Code",
      "authorId": 1
    },
    "2": {
      "id": 2,
      "title": "Clean Architecture",
      "authorId": 1
    }
  },
  "users": {
    "1": {
      "id": 1,
      "email": "chano@amalgama.co",
      "nickname": "Chano",
      "favoriteBookIds": [1]
    },
    "2": {
      "id": 2,
      "email": "sebastian@amalgama.co",
      "nickname": "Biche",
      "favoriteBookIds": [1, 2]
    }
  },
  "authors": {
    "1": {
      "id": 1,
      "name": "Uncle Bob"
    }
  }
}
```


La lógica para realizar el guardado en el store la podrán encontrar en el extraReducer del state principal de la aplicación:

https://github.com/gsanchezzusaeta/amalgama/blob/main/src/redux/reducers/mainReducer.js

Luego en la Vista de Home se muestra como puedo acceder fácilmente a la data:

https://github.com/gsanchezzusaeta/amalgama/blob/main/src/routes/Home/Home.jsx

La estructura que se brinda permite normalizar la información que llega desde el BackEnd, almacenar la data de esta forma permite evitar duplicados, mayor simplicidad (no guardo el objeto entero de un libro favorito, solo una referencia) y también mayor escalabilidad y performance para mayor cantidad de información.