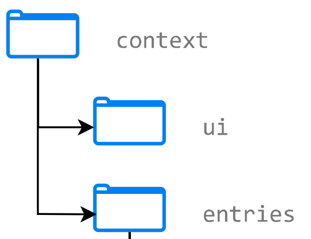

index.ts

```
export * from './EntriesContext'
export * from './EntriesProvider'
export * from './entriesReducer'
```

EntriesContext.tsx

```
import { createContext } from 'react';
import { Entry } from '../../interfaces';

interface ContextProps {
    entries: Entry[] ;

    // Methods
    addNewEntry: (description: string) => void
    updateEntry: (entry: Entry, showSnackbar?: boolean) => void
}

export const EntriesContext = createContext({} as ContextProps);
```

EntriesProvider.tsx

```
import { FC, useReducer, ReactNode, useEffect  } from 'react';
// const { v4: uuidv4 } = require('uuid');

import { Entry } from '../../interfaces';

import { EntriesContext, entriesReducer } from './';
import { Description } from '@mui/icons-material';
import { entriesApi } from '../../apis';

import { useSnackbar } from 'notistack';

export interface EntriesState {
    entries: Entry[];
}

const Entries_INITIAL_STATE: EntriesState = {
    entries: [],
}

interface EntriesProviderProps {
    children: ReactNode;
}

export const EntriesProvider: FC<EntriesProviderProps> = ({ children }) => {
    const [state, dispatch] = useReducer(entriesReducer, Entries_INITIAL_STATE)

    const { enqueueSnackbar } = useSnackbar();
const addNewEntry = async ( description: string ) => {
    // const newEntry: Entry = {
    //     _id: uuidv4(),
    //     description,
    //     createdAt: Date.now(),
    //     status: 'pending'
    // }

    const { data } = await entriesApi.post<Entry>('entries', {description})

    dispatch({ type: '[Entry] Add-Entry', payload: data });

}

const updateEntry = async ( { _id, description, status }: Entry,
    showSnackbar = false  ) => {

    try {
        const { data } = await entriesApi.put<Entry>(`entries/${_id}`,
            { description, status } );

        dispatch({ type: '[Entry] Entry-Updated', payload: data });

        if (showSnackbar)
            enqueueSnackbar( 'Entrada actualizada', {
                variant: 'success',
                autoHideDuration: 1500,
                anchorOrigin: {
                    vertical: 'top',
                    horizontal: 'right'
                }
            })

    } catch (error) {
        console.log({error});

    }


}

const refreshEntries =async () => {
    const {data} = await entriesApi.get<Entry[]>('./entries');
    dispatch({ type: '[Entry] Refresh-Data', payload: data})

}

useEffect(() => { refreshEntries(); }, [])


return (
    <EntriesContext.Provider value={{
        ...state,

        // Methods
        addNewEntry,
        updateEntry,
    }} >
        { children }
    </EntriesContext.Provider>
)
```

```
    )
}
```

entriesReducer.ts

```
import { Entry } from '../../interfaces';
import { EntriesState } from './';

type EntriesActionType =
| { type: '[Entry] Add-Entry', payload: Entry }
| { type: '[Entry] Entry-Updated', payload: Entry }
| { type: '[Entry] Refresh-Data', payload: Entry[] }


export const entriesReducer = ( state: EntriesState,
    action: EntriesActionType ): EntriesState => {

    switch (action.type) {
        case '[Entry] Add-Entry':
            return {
                ...state,
                entries: [ ...state.entries, action.payload ]
            }

        case '[Entry] Entry-Updated':
            return {
                ...state,
                entries: state.entries.map( entry => {
                    if ( entry._id === action.payload._id) {
                        entry.status = action.payload.status
                        entry.description = action.payload.description
                    }
                    return entry;
                })
            }

        case '[Entry] Refresh-Data':
            return {
                ...state,
                entries: [...action.payload]
            }

        default:
            return state;
    }

}
```