Flog "provider" (mulitple sources)

UI

Flog source

Participants: User | <DropBoxFlogs> | <Entry> | <EntryList> | <AddEntry> | <OpenFlogs> | useFlogs | useDropboxFlogs | useDropboxFiles | DropBox SDK

User → <DropBoxFlogs>: Selects flog file to open

composable function: **openFlog()**

> *the openFlog() function offerend by useFlogs uses the flog source APIs to retrieve the flog data, and load it for use in the openFlogs ref*

useFlogs → useDropboxFlogs: **loadFlogEntries()**

useDropboxFlogs → useDropboxFiles: **loadFileContent()**

useDropboxFiles → DropBox SDK: **checkAndRefreshAccessToken()**

useDropboxFiles → DropBox SDK: **filesDownload()**

DropBox SDK → useDropboxFiles: *callback:* **content (raw file text)**

useDropboxFlogs: **deserializeFlog updates flog**

useDropboxFlogs → useFlogs: **loaded entries are now in the useFlogs openFlogs ref**

props: **entryValue: (starter value as an EntryData object with both date and entry string)**

composable refs: **openFlogs**

> *useFlogs provides a ref object of all openFlogs, which is manages in-memory.*
>
> *It also provides an API of functions for updating the flogs and saving them back to source, and loading them from source.*

<OpenFlogs> → <EntryList>: *props:* **entries: (array of IEntry objects), editingEntry: (single IEntry object), readOnly: boolean**

> *<OpenFlogs> uses the openFlogs ref to populate its UI sub-components*

<EntryList> → <Entry>: *props:* **entry: (IEntry object),**

> *The <Entry> renders the date and string data to the UI*

<AddEntry> → <EntryList>: *props:* **entries: (array of IEntry objects), editingEntry: (single IEntry object), readOnly: boolean**

User → <AddEntry>: Submit

<AddEntry>: **submit handler submitAdd()**

<AddEntry> → <OpenFlogs>: **emits newEntry**

<OpenFlogs>: **newEntry handler addNewEntry()**

<OpenFlogs> → useFlogs: **addEntryToFlog()**

> *The entry is added to the flog, which is in a collection of open flogs, which is managed by useFlogs in-memory*

<OpenFlogs> → useFlogs: **saveFlogToSource()**

> *The flog is saved back to source*

useFlogs → useDropboxFlogs: **saveFlogEntries()**

> *The list of entries for the flog is updated and reactively updated in the UI, trickling down to child componenets*

> *The flog file is saved back to source*

useDropboxFlogs → useDropboxFiles: **serializeFlog saveFileContent()**

> *The flog file is saved back to DropBox*

useDropboxFiles → DropBox SDK: **checkAndRefreshAccessToken()**

useDropboxFiles → DropBox SDK: **filesUpload()**

Annotations

The date+time is saved in a specific string format in the flog text file.

A regex pattern matching that date+time string format is used to split the text file into frontmatter and entries.

The date+time string is deserialized into a Date object property of an Entry object when the flog entries are loaded from file.

And the Date object is serialized back into a string when the flog is saved back to file.

The <Entry/> component displays the date+time. If displaying it as a string (rather than a calendar+clock, for example) it needs to format the Date object as a string.

This is a good separation because useDropBoxFiles needs to own the date+time format used in the file, and the logic for converting to/from it.

The Entry object decides how the date+time is displayed to the user, which may or may not match the format saved in the file. Maybe it's displayed according to user preferences. Maybe it's displayed using a third party date UI component, which will probably take a Date object prop.

And the EntryList component might allow sorting or filtering entries by date+time, in which case it needs the date+time as Date objects.