The Firebase Firestore functions are very generic and can be used for multiple types of similar CRUD operations. However, as the codebase grew, some functions were written to perform specific operations.

1. The Firebase Firestore Core Functions

These pieces of code were written to generic ones and are the core of the Firestore operations; they were written to be reusable in multiple scenarios; they are found in `/app/api/firestore/index.ts` — this file contains all the core Firestore functions;

Examples of the generic functions are `addDoc` (which uses the Firestore `setDoc` function), and `getOneDoc` (which uses the Firestore `getDoc` function), examples of the more specific functions are `getManySearchDocs` (for returning items match from search query) and `addManyBookmarksToCollection` (for adding items to a collection).

2. The Firestore Hooks

We used Tanstack's React Query to utilize getting predictable states of each request in a clean and maintainable way; these hooks can be found in `/app/api/firestore/index.ts` — These hooks follow the React Hooks naming of prefixing "use" (eg. useEditDoc). They are written on top of the core functions mentioned above.

Examples of these hooks are `useGetDoc` (which is used to retrieve one document from Firestore) and `useEditDoc` (which is used to edit a document in Firestore).

3. Usage

The hooks are wrappers around the core functions that make working with these core functions easier; an example of usage can be seen below.

```
const CollectionDetails = FirestoreHooks.useGetDoc<IClipmateCollections>(
  ["collections"],
  params.id,
  {
    key: "getAllCollectionsDetails",
    placeholder: DetailsStore.store[params.id],
    gcTime: Infinity,
  }
);
```

Note: A peculiarity with how we use useQuery (Tanstack's Hook for data fetching), is that we pass in cached data to it using the `placeholder` option in the Tanstack useQuery options. This is done so that once the user has loaded data, the user will see some stale data before data is fetched.

Context Providers

There are various React Context providers that provide access to data or access to perform some global action; these providers are mentioned below with the action of data they provide. These providers are found in `/providers` and their context is found in `/context`, then their hooks are found in `/lib/hooks/index.ts` accordingly. Below are some of the more important ones;

1. `AccessControlProvider`: This context provider handles the access control of users that do not have an activation code; if a user is not activated it pops up a dialog box with a form to provide the activation code and proceed to using the application.

2. `CollectionsProvider`: This context provider handles the opening & closing of dialog, and creation of collections, it is written and placed in the React DOM tree in such a way that it is accessible from almost anywhere in the application.

3. `CommandKProvider`: This context provider is the most recent and handles the opening & closing of dialog, and action of the CMDK menu; it is placed at a point in the React DOM tree where you can perform a lot of actions.

4. `CopyPasteProvider`: This context provider handles the opening & closing of dialog, and saving of links, it is placed very high in the React DOM tree so that you can paste a link anywhere in the application using CMD + V (or Ctrl + V).

5. `DragNDropFromOSContextProvider`: This context provider handles the opening & closing of dialog and processing files that will be uploaded to Clipmate (right now only supports PDF), it handles the drag and drop from the OS (ie. drag a file from a folder on your computer).

6. `FullPostPopupProvider`: This is not a context provider per se, but a makes it possible to open and close the full post popup from any of the pages in the application.

7. `MultipleItemActionProvider`: This handles the multiple items selected and the actions performed on them; be it moving to trash, adding to a collection, removing from a collection, or restoring from trash as well.

8. `PaginatedPageProvider`: This context provider handles the provision and management of data on each page (paginated data from Firestore), there are variations of this for collections and sources pages.