Routing

There is a specific url route for every page and tab within a page in Atlantis. Each has a specific url route. When needed, we also have routes for specifc states of a view or tab.

For example, when viewing a document request you are taken to the document request page, with the Inbox tab selected, but the content that is shown is not the inbox but instead, the document request to be completed. There is also a third state of the inbox tab creating a new document request.

The url routes for each of these states in this example are as follows:

- /docrequests
- /docrequests/inbox
- /docrequests/new
- /docrequests/inbox/:id

Where is routing setup?

In the app.tsx file there is a LayoutRoutes function component. In it, all the different routes in the app are defined.

You will notice that we verify the logged in user has permission to go to the route before we actually add it. This ensures that they will be taken to a not found page in the case they are accidentally routed somewhere they shouldn't have permission to.

In the LayoutRoutes component routes have been grouped by the page they are on.

In the app component (also in app.tsx) you will also notice that we do not render the layout route components until the logged in users account settings and system permissions have been loaded to ensure routes are not rendered until we can validate if they have permission to navigate to a specific route.

RoutePaths.ts

In the RoutePaths.ts file there are many enums (and types) related to routing.

RoutePath type

The RoutePath type represents each and every route that can be navigated to in Atlantis. The RoutePath type is actually not an enum, though in usage it operates as one. It is a defined type and is actually a concatenation of many enum types. RoutePath is composed of an enum that represents the route of each page, unless the page doesn't have tabs in which case the route is part of an enum name SingleRoute. For example, we have a `DocumentRequestRoutePath` that currently looks like the following.

```
export enum DocumentRequestRoutePath {
    DocumentRequestIndex = '/docrequests',
    DocumentRequestInbox = '/docrequests/inbox',
    DocumentRequestNew = '/docrequests/new',
    DocumentRequestView = '/docrequests/inbox/:id',
    DocumentRequestTemplates = '/docrequests/templates',
    DocumentRequestTemplateNew = '/docrequests/newTemplate',
    DocumentRequestSent = '/docrequests/sent',
    DocumentRequestAccountRequests = '/docrequests/all',
}
```

You can see there is a route path the for the page itself, others for each tabs, and even others for states within these tabs.

Why is this helpful?

This allows us to constrain route paths when needed. For example, I can have a function that can accept RoutePaths of a specific page (like document requests). I don't have to worry about handling the other 50+ route paths that exist, only the document request route paths.

RouteParams

In the RoutePaths.ts file there are enums that end with RouteParams. These represent the variables in route paths that have them. For example, for the path '`/docrequests/inbox/:id'` there is a variable name ':id'. You can use DocumentRequestRouteParams.Id enum whenever you need to, so you don't have to have magic strings anywhere. This is especially important when routing (see How do I route to a specific location?).

Other enums

At the time of this writing, there is only one other type of enum. It is called `SelectedDetail` . It's purpose is to be used in conjuction with the `DocumentRoutePath.GoToNodeDetail` (/documents/node/:nodeId/:selectedDetail). The values of this enum are all the valid values that can be used for the `:selectedDetail` variable.

There will likely be other enums that serve a similar purpose (and maybe even other purposes) and feel free to add other enums and types to this file to make routing as clean as it can be.

How do I route to a specific location?

useRouting Hook

The main way to route to a specific location is with the useRouting hook. In a similar way to the RoutePath type being composed of other enums, it is a hook that is composed of other hooks. Most areas of the app have their own hook with all the logic for routing contained there. When actually using the hook though, you don't have worry about that and can just import useRouting and use what you need from it.

There are currently two ways to use the useRouting hook:

1. routeTo... functions
   - Use these functions when you need to route directly to a specific area
     - Example: `routeToDocuments();`
   - Some of these functions take parameters, like ids
2. getLinkTo... functions
   - Use these functions when you need to grab a link to an area, maybe for use with Component Library components
     - Example: `<Card.Link tag={Link} to={getLinkToMySettings()}>{t(NavMenuKeys.MySettings)}</Card.Link>`
   - You need to specify that the `tag` property is '*Link*', and the `to` is where you call the getLinkTo function.

useNavigate Hook

The majority of cases should be covered by the useRouting hook, but if you come across a case where that won't work, you can still use the useNavigate hook:

- navigate(RoutePath.RouteWithoutVariable) - route that does not have a variable
- navigate(RoutePath.RouteWithSingleVariable.replace(SomeRouteRouteParams.paramName, valueToReplaceVariableWith);
- navigate(
      RoutePath.RouteWithMultipleVariable
          .replace(SomeRouteRouteParams.parameName1, param1Value)
          .replace(SomeRouteRouteParams.parameName1, param1Value)
  );

How do I add a route?

- Add a new enum to the `RoutePath` type.
- In the `canGoToRoute` function in the `useRouteValidator` hook, add a case to the switch statement to validate the logged in users permissions and features appropriately to make sure they can navigate to this route.
- In the LayoutRoutes component, add a Route component for this specific route. Make sure you only render the route if the logged in user has permission. For example: `{canGoToRoute(RoutePath.AuditLogs) && <Route path={RoutePath.AuditLogs} element={<AuditLogs />} />}`
- If needed (like on a page with tabs), make sure the component that you've chosen to render when this route is navigated to, has been updated to handle this route.
  - For examples of this, look at the code for any component that has tabs.

- Add a getLinkTo and routeTo function for the route you added
    - If the route you added is its own new area, consider making a new hook for it within the Routing folder and then make sure to add it to the general useRouting hook in the return statement

---

Revision #6
Created 19 January 2023 04:10:33 by Quinn Godfrey
Updated 1 August 2023 17:45:52 by Josh Reese