Report on

Refactoring Duplicated Components in the SIGMA Project Developed by ReactJS

Second Step: Routes Handling Hook

Related to the tasks SIG-126, MOJ-202 and MOJ-203

Mohammad Mollaahmadi
MOJ Secure Company
Tehran, Iran
Aug 2024

Introduction

The SIGMA project, developed using the ReactJS library, currently contains multiple instances of duplicated components. This redundancy increases maintenance efforts and can lead to inconsistencies across the codebase. We recently reviewed the project's current state and devised a three-step plan for refactoring: the API calls module, the routes handling hook, and the screen components.

This document discusses our approach to addressing the second section: the routes handling hook, and presents the results of our changes. This report focuses on refactoring the current routes handling hook by dividing it into separate sections: documents, attachments, and cartable. We have established new rules for creating routes for different pages in the project and developed new route handling based on these guidelines. Each of these sections will be discussed in detail.

Implementation

Step 1: Establish a set of comprehensive rules

As part of the first step, we have established a set of comprehensive rules for routing the cartable, which includes the modules in the business such as import, export, transit, gate, and temporary export. These rules aim to standardize the way routes are managed across the project, ensuring consistency, scalability, and ease of maintenance:

{module-slug}/cartable/[user-roles]: This route handles the cartable for different user roles within each module.

{module-slug}/documents/{record-id}: This route provides access to documents associated with a specific record ID within a module.

{module-slug}/documents/{record-id}/edit: This route ensures consistency in accessing and editing documents within a module based on a specific record ID.

{module-slug}/documents/{record-id}/attachments/{attachment-type}/{id}: This route manages attachments related to a specific record, categorized by attachment type and ID.

{module-slug}/documents/{record-id}/attachments/{attachment-type}/{id}/subType: This route manages a type related to an attachment associated with a specific record, categorized by subtype, attachment type, and ID.

Definitions:

- **id and record-id**: These are identifiers for a document, consisting of a combination of numbers and letters. They uniquely identify each document within the system.
- **user-roles**: This parameter specifies the role of the user accessing the route. Although this variable does not impact the components for now, it clarifies the route and is also received from the response of the `/Menu` API. The roles include:
    - **expert**
    - **evaluator**
    - **customs-manager**
    - **tariff-determination**
    - **determining-value**
    - **technical-assistant**
    - **review**
- **module-slug**: This represents the specific module within the business and includes the following:
    - **import**
    - **export**

- **transit**
- **returned**
- **temporary-export**
- **financial**
- **gate-management**
- **gate**
- **customers**
- **system-management**
- **review**

NOTE: Regarding **gate** and **gate-management**, it is important to note that these are different modules. The gate module pertains to the cartable, whereas gate-management is a distinct module.

By implementing these rules, we aim to create a more organized and maintainable routing structure for the cartable, facilitating easier navigation and better management of the modules in our business operations.

Step 2: Modify the Response of /permissions/Menu API

In addition to defining new rules for routing, we need to update the response of the /Menu API to align with these rules. Below is a list of changes required for the API response:

1. **Rename `moduleName` Field**:
   - Change the `moduleName` field to `persianModuleTitle` as it is only used to display a tooltip.
2. **Add `moduleSlug` Field**:
   - Introduce a `moduleSlug` field in each module object. This field will be used to identify the selected module, manage display changes, and select the first menu item in the first module by default after the page renders.
3. **Modify Path Structure**:
   - Standardize the `path` in each item to ensure consistency across all path structures.

By implementing these modifications, we aim to ensure that the API response is consistent with the new routing rules, facilitating better management and display of modules.

Step 3: Modify the Custom Hook for Handling Routes and Develop New Modules

After updating the routing rules and modifying the response of the `/Menu` API, the next step is to create a custom hook to manage the routes. This involves reorganizing the project structure and developing new modules to handle cartable, documents, and attachments routes separately.

1. **Create a New Directory**:
    - Make a new directory called `routes` in the `src` folder.
2. **Move Existing Files**:
    - Move all existing route-related JavaScript files into the new `routes` directory.
3. **Develop New Modules**:
    - Create new modules for handling specific routes: `attachmentsRouteManager.jsx`, `cartableRouteManager.jsx`, and `documentsRouteManager.jsx`.
4. **Update Project Structure**:

Ensure the final directory structure is as follows:

./src/routes

├── attachmentsRouteManager.jsx

├── cartableRouteManager.jsx

├── documentsRouteManager.jsx

├── dynamicIndex.jsx

├── menu-service-mock.json

├── RouteMap.js

├── routes.jsx

└── useRoutesCustom.jsx

By implementing these steps, we will create a well-organized and modular routing structure, making it easier to manage and maintain the different route handlers for cartable, documents, and attachments.

Conclusion

In this effort to enhance our project's routing structure, we have undertaken several key steps to ensure a more organized, scalable, and maintainable system.

1. **Establishing Routing Rules**: We defined a comprehensive set of routing rules to standardize the handling of various modules, such as import, export, transit, gate, and temporary export. These rules aim to create consistency and ease of navigation within the project.
2. **Modifying the /Menu API Response**: To align with the new routing rules, we updated the response of the /permissions/Menu API. This included renaming fields, adding new fields, and standardizing path structures to ensure better integration and functionality.
3. **Developing a Custom Hook and New Modules**: We created a new directory structure to house our route management files, developed new modules for handling specific routes, and designed a custom hook to manage these routes effectively. This modular approach enhances the flexibility and maintainability of our routing system.

By implementing these changes, we have significantly improved the project's routing framework. This will facilitate easier navigation, enhance the user experience, and make future maintenance and updates more straightforward. Our efforts ensure that the project is well-equipped to handle the complexities of managing different modules and their respective routes efficiently.