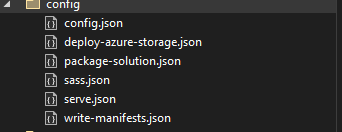
**Pages Display Webpart Documentation**

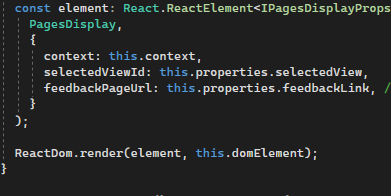
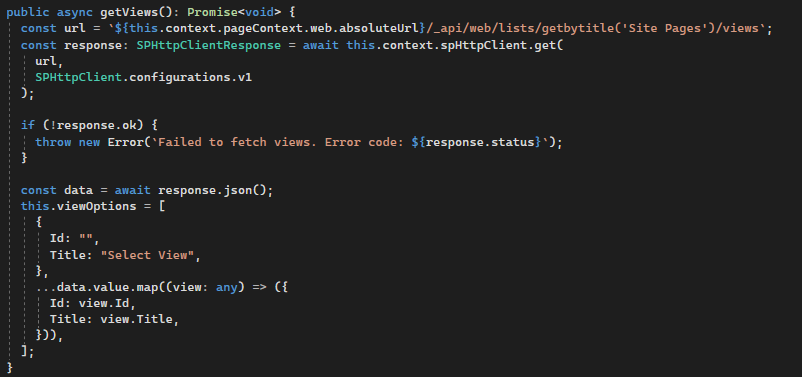
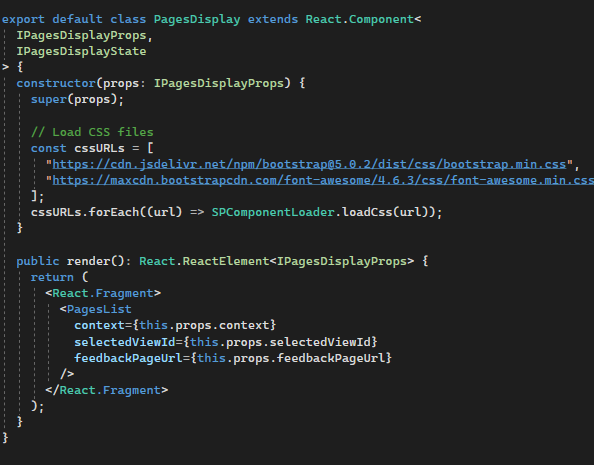
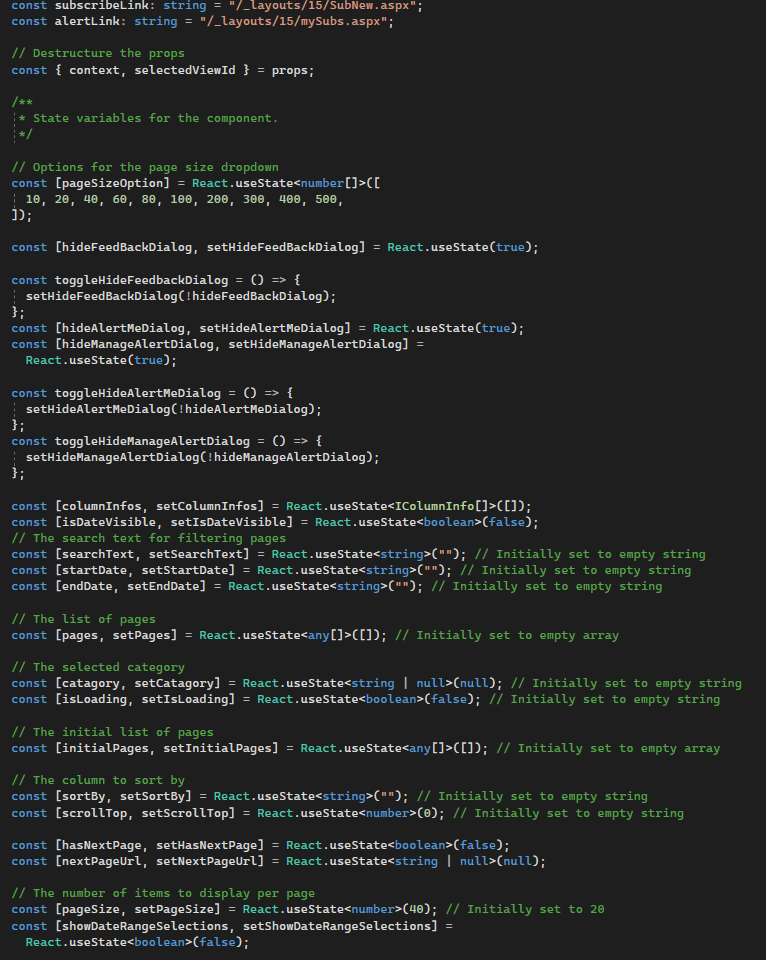
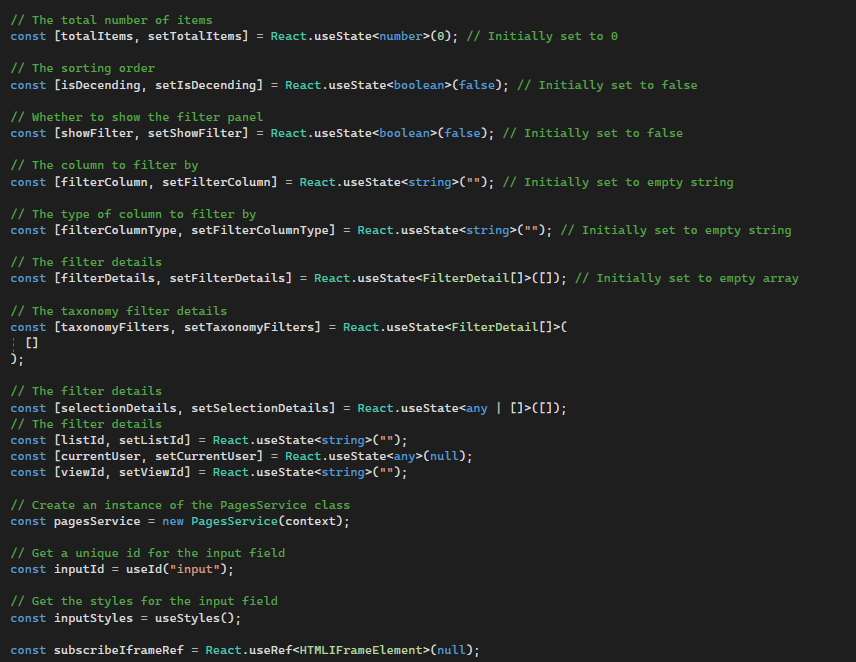
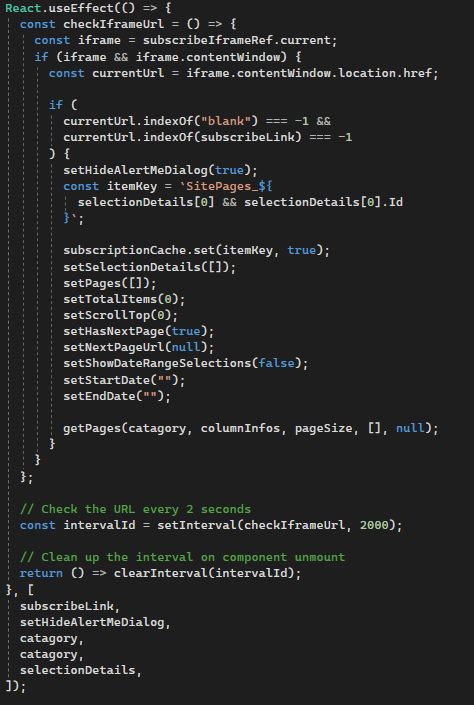
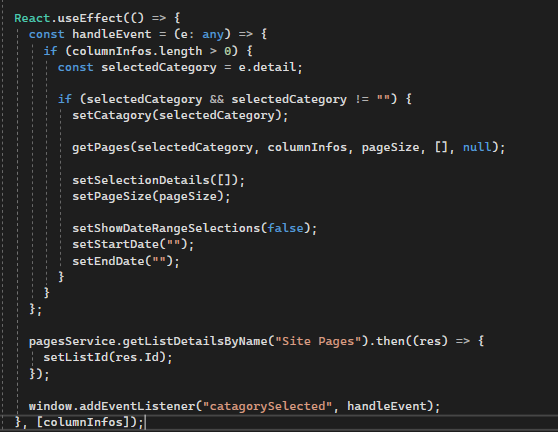
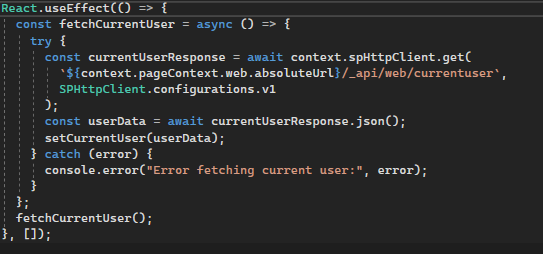
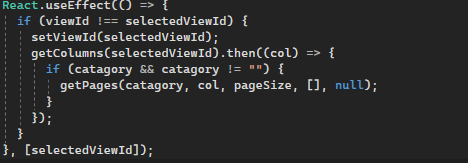
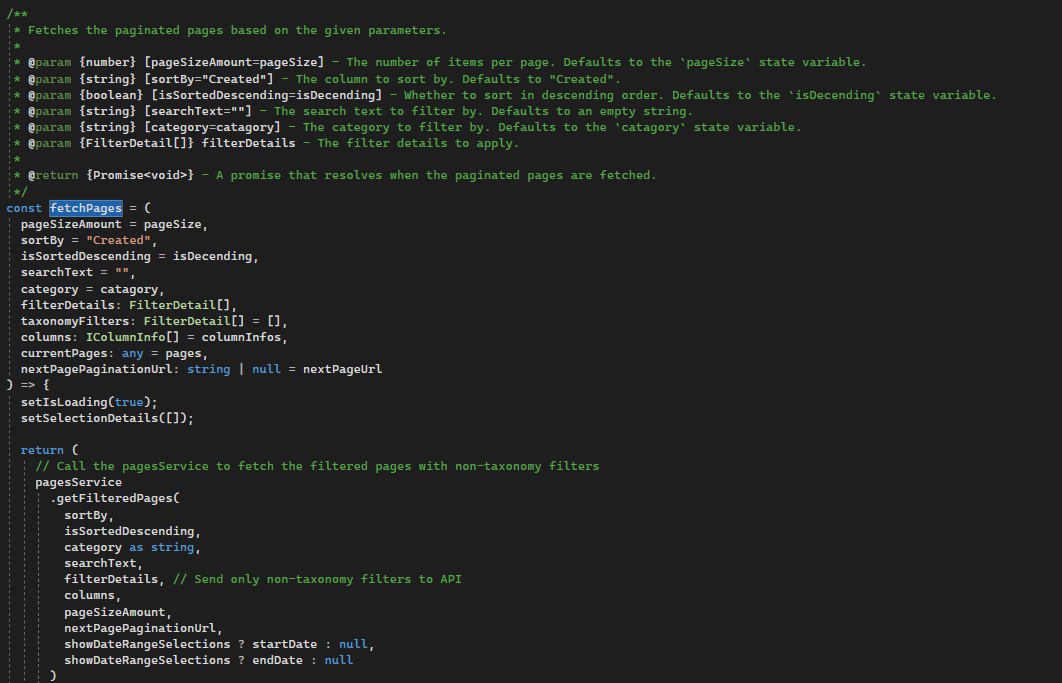
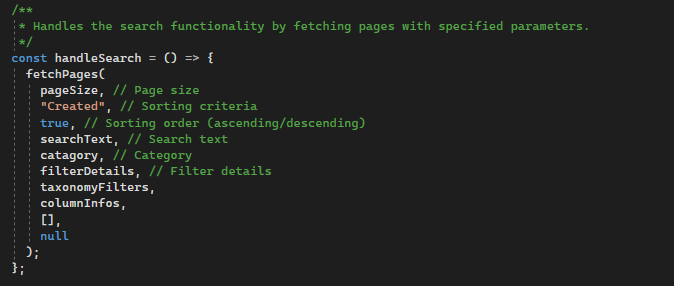
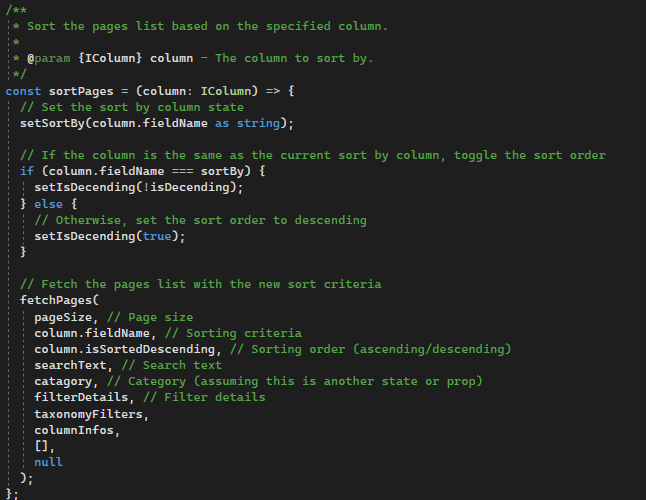
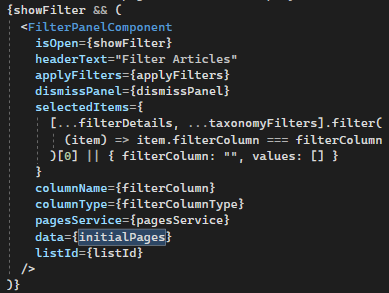
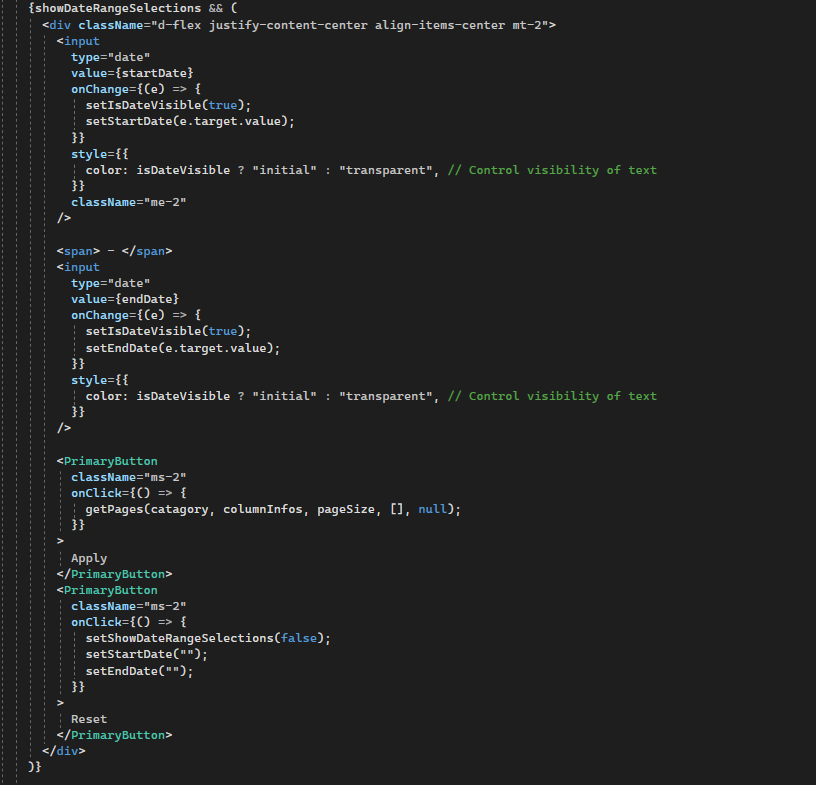
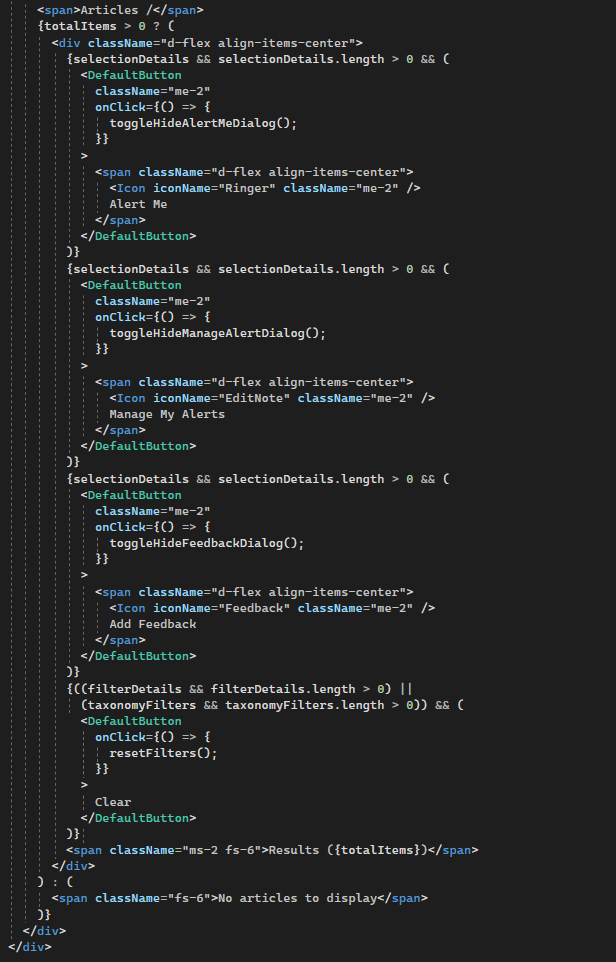
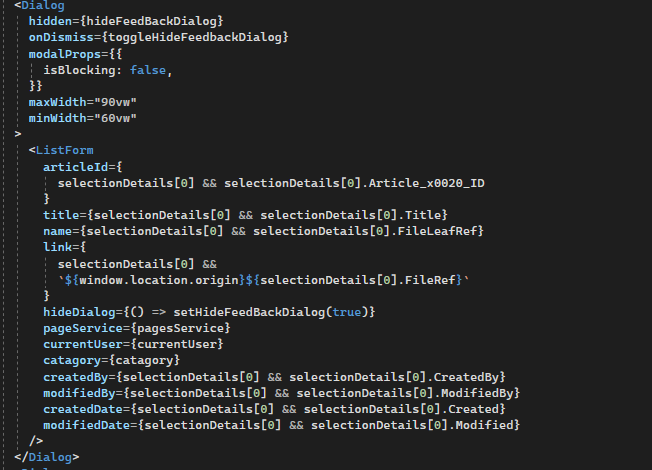
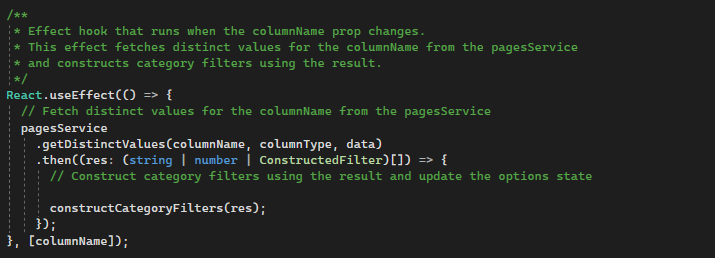
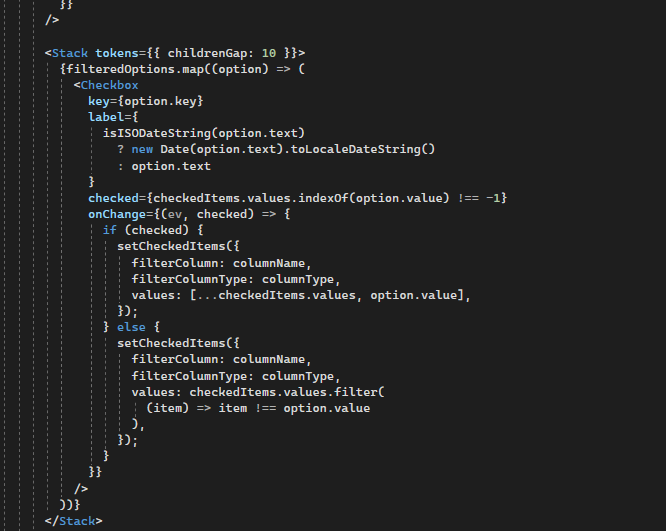
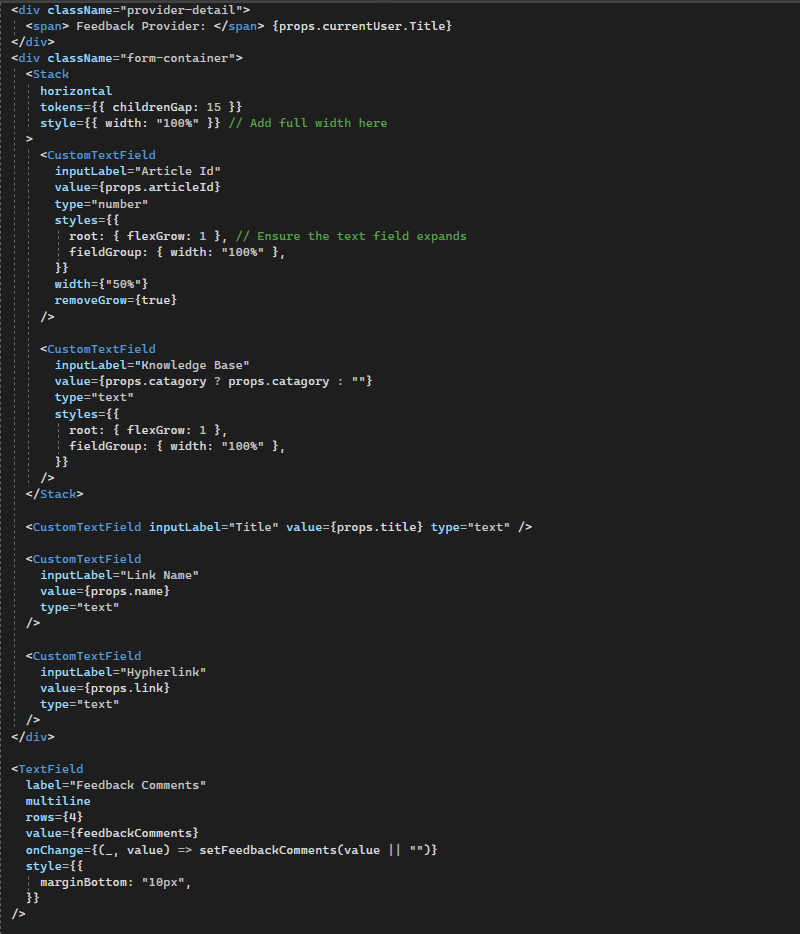
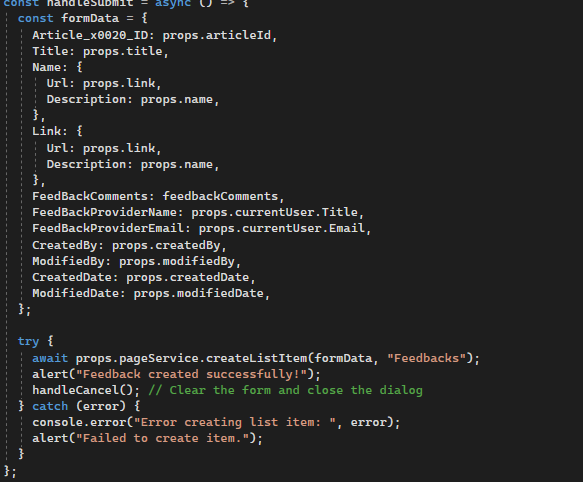
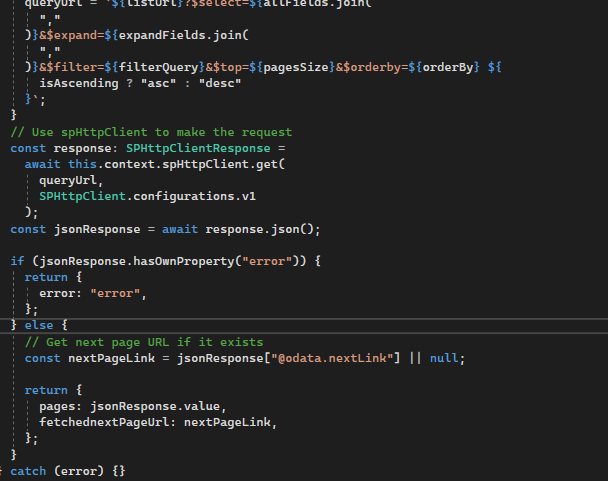
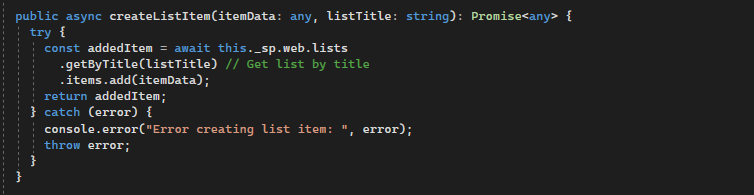
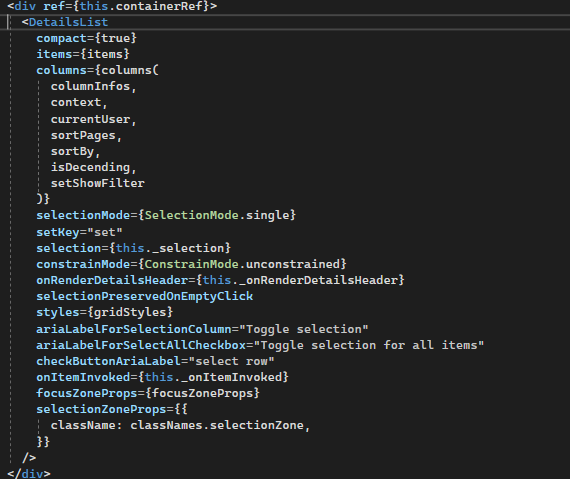
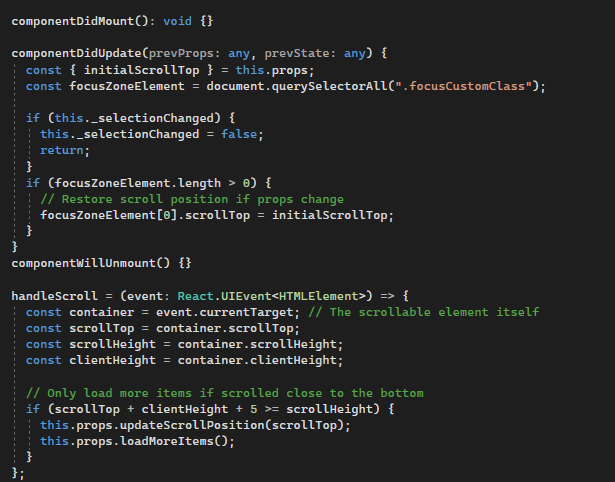
**Development environment setup** - You can follow this instruction from [link](https://learn.microsoft.com/en-us/sharepoint/dev/spfx/set-up-your-development-environment) .

1. **Node js** - Install the lts stable version of Node js using [link](https://nodejs.org/en) .
2. **Gulp cli** - For running tasks such as clean, serve and bundle.   
   - npm install gulp-cli --global - run this command after installing node.
3. **Yeoman** - Generates boilerplates for different types of projects  
   - npm install yo --global
4. **Yeoman sharepoint generator** - Helps in creating a sharepoint client side project  
   - npm install @microsoft/generator-sharepoint --global
5. **Gulp Trust** - Trust certificates by running - gulp trust-dev-cert

Gulp tasks

1. **gulp-serve** - Serving the project to local workbench  
   Make sure you have added your tenant inside the serve.json file located under config folder.  
     
     
   After opening the folder modify the initial page to your workbench  
   
2. **gulp-bundle –ship** - Creating a production bundle by running this command
3. **gulp-package-solution –ship** - Creating a .sppkg file that will be used in side the app catalog for deployment.
4. **gulp-clean** - If there is a previous deployment available we have to clean it in order to optimize the size of the package

**Code structure and folders**

1. **PagesDisplayWebpart.ts** -   
   Entry file for the project. This is where the project context lies; we can also define the configuration property pane controls.  
     
     
   There are two property configuration and the values of the selected items is stored under the “selectedView” and “feedbackLink” variables we are then passed to the PagesDisplay component.   
     
     
     
     
     
     
     
     
   The view options are also fetched under this component  
     
     
   The variable viewOptions is what we used for the selectview drop down on the previous screen shot.
2. **PagesDisplay.tsx** - Component  
   This is where we are adding different css from a cdn into our project.   
     
     
   We are also passing the variables to the PagesList.tsx component.
3. **PagesList.tsx**   
   This component is where our entire logic is stored.  
     
     
     
     
   In the above two screenshots, we can find all the variables that we are using to store data. For example, the pages state is where we store all the fetched pages.  
   The pagesSizeOption is where we store the list of number options and so on.  
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   Inside this file there are 4 useEffect hooks.  
     
     
   This useEffect is responsible for resetting and also fetching items when there is a change in the alertMe url. Since the alert me is embedded inside the project using Iframe, we can only close the popup when there is change in the url.  
     
     
     
   This useEffect runs is responsible for assigning an event listener. The side nav or the term store navigation will fire an event called “catagorySelected” and this pagesDisplay webpart will listen to that event and get what the current selected KnowledgeBase is. So when ever we select a knowledge base the handleEvent function will execute which inturn has some different data fetching and state resets.  
     
     
     
   This useEffect only runs at the start and fetches the currentUser details which will be used for the FeedBack form and so on.  
     
     
   This useEffect runs when there is a change in the view, which can be done by updating in the property pane.  
     
   **Data fetching**  
   A. For data fetching I have created another file called PagesService(will be described in the later sections) and we will utilize that service   
     
     
     
     
   Depending on any parameters passed the service will provide a data. So when ever we apply a search, sort, filter, page and pageSize change we will provide different parameters to this function and it will fetch the data and update our state.  
     
   For example - **Search**  
     
     
   Here we provided the search text to the function.  
     
   **Sort**  
     
   Here we provided the sort details. And so on.   
     
     
   **Render** - On the render I have placed different components that will be visible depending on the variables stated earlier.  
     
   Shows the **FilterPanelComponent** when **showFilter** is true.  
     
     
   Shows the date controls at the top depending on whether showDateRangeSelections state is true.  
     
     
   Shows the item actions by checking if there is a selected item.  
     
   This renders the list of pages that are being passed as items. The component also receives different props that will help in rendering such as initialsScrollTop and so on.  
     
     
   This dialog is rendered depending on whether the “Add feedback” button is clicked, which will in turn update the state “hideFeedBackDialog” to false. Here we are also passing some properties that are going to be a default value for the form such as articleId, title, currentUser and so on.
4. **FilterPanelComponent** - A component that is responsible for rendering the side panel that is drawn when we click on the filter icons next to the columns.  
   Since there is no API to fetch the list of unique values for a column I am constructing the options by receiving the list of data and the column name. There is a method under the PagesService file that will help us in getting the distinct or unique values on a column.  
     
     
     
   We will the use the constructed filters  
     
   And when an apply is selected we will call the method “applyFilters” from the parent component “PagesList” as mentioned previously.
5. **ListForm.tsx** - The form for the add feedback. The code contains the list of inputs for the feedback.  
     
   When submit is clicked it will execute the following method.  
     
   We are using another method called “createListItem” from “PagesService”.
6. PagesService.ts - The file or service where all api calls and data restructuring is placed. The services there are descriptive.  
     
   getFilteredPages - Receives different parameters that are going to be used for constructing the request url for the sharepoint api.  
     
     
   After constructing the filters we will use the spHttpClient service that is by default available in spfx project to execute our rest api call. The “queryUrl” contains the final url to be executed and it will contain the filter the order by and other parameters.  
     
   CreateListItem - This method is used in creating an item inside a list. This is the method we used for submitting the feedback.  
   
7. ReusableDetailList.tsx - This component is responsible for rendering the pages and it uses the DetailList component from FluentUi.  
     
   I have passed different configuration and data for the detail list component from fluent Ui. And it will render accordingly.   
     
   These life cycles are responsible for updating the current scroll position to the place where the client was scrolling previously and also loading more items when the user is near the end of the scroll by calculating the scroll height and client height.   
     
   The above are the main components responsible for the PagesDisplayWebpart. Other helper components are clear and have no complicated logic under them such us ColumnDetails (responsible for customizing how the header and also the cells are displayed) and columnUtils ( responsible for providing a minimum and maximum width of a column).